VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

ANDROID APLIKACE PRO VYČÍTÁNÍ DAT Z CHYTRÝCH MĚŘIČŮ POMOCÍ USB SENZORU A NÁSLEDNÉ ODESÍLÁNÍ DAT DO CENTRÁLNÍ DATABÁZE

APPLICATION FOR SMART METER DATA READOUT WITH SUBSEQUENT SENDING TO THE CENTRAL DATABASE

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. Tomáš Navrátil**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Kryštof Zeman**

SUPERVISOR

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Tomáš Navrátil ***ID:*** 143931
Ročník: 2 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Android Aplikace pro vyčítání dat z chytrých měřičů pomocí USB senzoru a následné odesílání dat do centrální databáze

POKYNY PRO VYPRACOVÁNÍ:

V dnešní době probíhá intenzivní vývoj v oblasti mobilních zařízení a jejich zapojení v průmyslu. Jako jedna z možností se nabízí využití interního USB portu v mobilním telefonu. Cílem práce bude vytvoření Android aplikace pro měření fyzikálních veličin s využitím zařízení připojeného přes USB OTG. Tato data budou následně odeslána do centrální databáze.

DOPORUČENÁ LITERATURA:

[1] Develop. Android Developers [online]. [cit. 2016-09-14]. Dostupné z: https://developer.android.com/develop/index.html

[2] The basics of USB device development using the Android framework. Embedded [online]. [cit. 2016-09-14]. Dostupné z: http://www.embedded.com/design/connectivity/4437620/1/The-basics-of-USB-device-development-using-the-Android-frameworkThe-basics-of-USB-device-development-using-the-Android-framework

Termín zadání: 1.2.2017 ***Termín odevzdání:*** 24.5.2017

Vedoucí práce: Ing. Kryštof Zeman
Konzultant:

doc. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Tato práce se zabývá možnostmi využití přenosných zařízení, především mobilních telefonů a tabletů podporujících technologii USB OTG, pro odečítání údajů z chytrých elektroměrů. K tomuto účelu byla v praktické části práci vyvinuta aplikace pro OS Android schopná komunikovat s elektroměry pomocí standardu DLMS/COSEM za využití optického senzoru a sériové linky. Získaná data jsou následně ukládána do vzdálené databáze pro další využití.

KLÍČOVÁ SLOVA

Android, DLMS, COSEM, M2M, IoT, elektroměr, chytrá domácnost

ABSTRACT

This thesis explores new posibilities of using mobile devices, specificaly cell phones and tablets which support the USB OTG feature, to read data from smart electricity meters. To achieve this goal, an application for OS Android was developed in practical part of this thesis. This mobile application communicates with electricity meters via optical sensor serving as serial line and using the DLMS/COSEM standard. Acquired data are saved to a remote database for further use.

KEYWORDS

Android, DLMS, COSEM, M2M, IoT, electricity meter, smart home

NAVRÁTIL, T. Android Aplikace pro vyčítání dat z chytrých měřičů pomocí USB senzoru a následné odesílání dat do centrální databáze. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2017. 43 s. Vedoucí diplomové práce Ing. Kryštof Zeman.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Android aplikace pro vyčítání dat z elektroměrů přes USB sensor a následné odesílání dat do centrální databáze“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne ....

(podpis autora)

SIX research centre
sensor, information and communication systems

Faculty of Electrical Engineering
and Communication

Brno University of Technology
Technicka 12, CZ-61600 Brno, Czechia

http://www.six.feec.vutbr.cz

Výzkum popsaný v této diplomové práci byl realizovaný v laboratořích podpořených projektem Centrum senzorických, informačních a komunikačních systémů (SIX); registrační číslo CZ.1.05/2.1.00/03.0072, operačního programu Výzkum a vývoj pro inovace.

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

2007–13
OP Výzkum a vývoj pro inovace

PODĚKOVÁNÍ

Na tomto místě bych chtěl poděkovat panu Ing. Kryštofovi Zemanovi za jeho trpělivost, ochotu a cenné rady při psaní této práce, které mi pomohly k jejímu úspěšnému dokončení.

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Neustále se navyšující počet M2M (Machine to Machine) zařízení a jejich nároků pohání vývoj telekomunikačních sítí neustále kupředu. Některá zařízení zůstávají skrytá jako tzv. zabudované (embedded) systémy, další se poté stávají součástí infrastruktury nazývané jako Internet věcí (IoT). Tato zařízení komunikují prostřednictvím IP sítí a jejich počet se meziročně zvyšuje o 30 % [1]. Takto vysoký nárůst naznačuje, že v roce 2020 bude připojeno mezi 20 a 30 miliony zařízeními [2]. Kromě mobilních telefonů se jedná i o různé prvky tzv. chytrých domácností, především různých čidel, senzorů a měřících zařízení. Některá z těchto M2M zařízení však nemohou být z různých důvodů zahrnuta do IoT. Příčinou mohou být například bezpečnostní požadavky pro přenášení citlivých dat nebo nedostupnost síťové konektivity na místě fyzického umístění zařízení. V těchto případech je nutné hledat jiné metody komunikace.

Alternativním způsobem komunikace s M2M zařízením se zabývá i tato diplomová práce. Je v ní rozebráno a realizováno odečtení údajů z chytrého elektroměru, který je vybaven pro komunikaci přes IP sítě i pro lokální komunikaci přes sériovou linku. Tato komunikace je realizována pomocí DLMS/COSEM protokolu, který byl navrhnut především pro chytrá měřící zařízení.

Cílem práce bylo zjistit možnosti a komunikační strukturu DLMS/COSEM protokolu (kapitola 2) a tyto znalosti použít při vývoji aplikace pro mobilní zařízení podporující operační systém Android (kapitola 3.5). Toto zařízení je k elektroměru připojováno pomocí USB OTG optického senzoru, který se chová jako sériová linka a jsou přes ni z elektroměru získávána požadovaná data. Veškerou komunikaci řídí vyvinutá aplikace, která data následně i zpracovává. Finálním krokem je uložení získaných dat do centrální databáze.

V praktické části této práce byla vyvinuta aplikace schopná navázat komunikaci s elektroměrem a odečíst z něj identifikační údaje. Pro uložení dat do databáze byla zprovozněna serverová část sestávající z databázového softwaru MariaDB a komunikačního API.

1 INTERNET OF THINGS A MACHINE TO MACHINE

Machine to Machine (M2M) je pojem označující technologie umožňující komunikaci mezi elektronickými zařízeními bez vnějšího zásahu člověka. Na základě této komunikace jsou zařízení schopna vykonávat specifické operace.

Internet of Things (IoT) označuje systém elektronických zařízení komunikujících přes internet a využívajících principů M2M.

1.1 Historie IoT a M2M

Filozofie přímé komunikace mezi zařízeními je rozvíjena již od počátku 20. století, kdy však termín M2M neexistoval. Za první vynález využívající principů M2M je považován letecký radar (30. léta 20. století) a jeho nástavba využívající transpondér v podobě identifikačního systému IFF (Identify: friend or foe, 70. léta) pro detekci nepřátelských vojenských jednotek. Tyto systémy však byly těžkopádné a s vysokými náklady na výrobu, logistiku i provoz a proto nebylo jejich rozšíření a další adaptace mimo armádní sektor ekonomicky únosné.

Za první historický milník pro M2M technologií je tak považováno rozšíření mikroprocesorů, které umožnilo miniaturizaci koncových zařízení a tedy jejich snadnější rozšíření s ohledem na vývoj i logistiku. Tento významný technologický pokrok umožnil vznik technologií, které jsou nazývány základními pilíři moderních M2M systémů – RFID (Radio-frequency identification), telematika, telemetrie, senzorové sítě, chytrá zařízení a vzdálené monitorování. Právě využití technologie RFID (využívá konceptu IFF systémů), v logistice je považováno za jedno z prvních široce rozšířených využití M2M. Pohyb zboží označeného RFID tagem přes bránu nakládací rampy opatřenou RFID senzory s krátkým dosahem (a tedy s nízkým odběrem elektrické energie) umožňovaly automaticky měnit počet dostupného zboží na skladě. I když je toto využití RFID technologie považováno za ukázkový příklad M2M komunikace, jedná se pouze o velmi lokální aplikaci [3].

Za druhý milník ve vývoji M2M je označováno rozšíření Ethernetu a TCP/IP (Transmission Control Protocol/Internet Protocol) sítí, především Internetu. Standardy sady IEEE 802.3 poskytly předpis pro jednotnou formu komunikace mezi individuálními zařízeními, čímž bylo docíleno vzájemné kompatibility mezi produkty různých výrobců. Možnost kombinace různých výrobků dle potřeb zákazníků umožnilo další rozšíření M2M zařízení do dalších průmyslových odvětví i do soukromého sektoru [4].

Právě rozšíření M2M do soukromého sektoru je považováno za počátek éry Internetu věcí (IoT). Filozofie IoT je postavena na rozšiřování tzv. chytrých zařízení, jejichž základem jsou přístroje každodenního života, avšak obohacené o prvky technologií M2M. IoT není možné považovat za samostatné tržní odvětví nýbrž jako nástavbu odvětví již existujících, ve kterých poskytují výhody oproti dřívějším řešením s důrazem na ekonomickou životaschopnost a komfort obsluhy. Rychlého rozšíření se IoT zařízení dočkala především ve zdravotnictví, dopravě a energetice. Zdravotnickému personálu byl

například umožněn dohled nad pacienty v domácí péči a díky agregaci dat je tímto způsobem možné monitorovat více pacientů jedním zdravotníkem. Další zajímavou možností je sledování zdravotnického vybavení při pohybu po budově nemocnice nebo kontrola manipulace s léčivými, případně nebezpečnými látkami. Pro logistiku byla průlomovým okamžikem možnost online sledování polohy vozidel, což mělo za následek snížení nákladů na plánování tras řidičům jak spedičních společností, tak například i taxislužeb. Řidičům je nabízena řada doplňkové výbavy zahrnující navigační, zábavní a bezpečnostní systémy – například automatické kontaktování záchranných složek v případě dopravní nehody.

Nejnovějším odvětvím, do kterého se IoT zařízení rozšířila, jsou tzv. Chytré domácnosti. Obyvatelům takto vybavených domácností je umožněno ovládání elektrického vybavení a spotřebičů pomocí centrální jednotky lokálně z budovy i vzdáleně přes internet. Uživatelské rozhraní pro ovládání zařízení má formu webové stránky nebo aplikace pro mobilní zařízení a je přizpůsobeno pro méně zdatné uživatele. Kromě základního ovládání zařízení je umožněno i nastavení určitého stupně automatizace, např. předprogramování vytápění objektu nebo i jednotlivých místností [4], [5], [6].

1.2 Specifické vlastnosti M2M

Základní vlastností M2M zařízení je jejich maximální možná přizpůsobitelnost pro konkrétní použití. To těmto zařízením umožnilo v relativně krátkém období expandovat do nejrůznějších odvětví lidské činnosti, jak již bylo nastíněno v předchozí kapitole (1.1). Vlastnosti těchto zařízení vycházejí primárně z jejich praktického využití, a proto není možné popsat všechny jejich vlastnosti. Je však možné se věnovat vlastnostem, které mají M2M zařízení společné. Jedná se především o metody komunikace mezi zařízeními, se kterými jsou úzce spojeny další vlastnosti jako např. výpočetní výkon nebo energetické požadavky daného zařízení.

1.2.1 Komunikace

Na fyzickém médiu pro přenos dat v případě M2M nezáleží. Zařízení mohou komunikovat pomocí kabelových i bezdrátových přenosových sítí. Možnost spolehlivě používat bezdrátové sítě je důsledkem struktury komunikačních dat typických pro M2M aplikace či odlišného způsobu pojetí komunikace např. v porovnání H2H (Human to Human) a M2M. Zařízení mezi sebou komunikují pomocí krátkých datových zpráv, jejichž velikost je udávána v jednotkách bajtů. Komunikace mezi zařízeními neprobíhá neustále, nejčastěji je podřízena dvěma základním modelům:

1. Dotaz – odpověď

   Jedno zařízení reaguje na příkaz druhého.

   Např. meteorologická stanice požaduje aktuální informace o směru a rychlosti větru z příslušného senzoru.

2. Reakce na naprogramovanou událost

   Odeslání zprávy je iniciováno nastáním naprogramovaného stavu.

Např. čidlo detekující otevření dveří o této skutečnosti informuje centrální jednotku, která následně zajistí rozsvícení domovního osvětlení na příslušném místě nebo vyvolá alarm. Za vnější vliv je také možné považovat časovač, který zajišťuje periodické odesílání dat ze senzoru.

Z těchto komunikačních modelů vyplývá skutečnost, že při komunikaci mezi M2M zařízeními je největší důraz kladen na spolehlivý přenos dat. Tyto požadavky se však diametrálně liší od požadavků, které jsou kladeny na H2H (Human To Human) komunikaci a podle kterých jsou aktuální datové sítě vybudovány. Tyto sítě byly koncipovány pro použití technologie Ethernet (IEEE 802.3) a IP protokolu, kde datová komunikace podléhá principu „Best effort" a doručení odeslaných dat tedy není zaručeno. Důraz je zde naopak kladen na rychlost datového přenosu (desítky megabajtů za sekundu), což zajišťuje rychlé odbavení velkého množství dat a malá zpoždění (jednotky milisekund) při oboustranné komunikaci. Pro IoT se jedná o přijatelný kompromis a k zajištění kvality služeb je využíváno mechanismů protokolu TCP (Transmission Control Protocol) nebo jsou datové zprávy řízeny aplikační vrstvou. Z pohledu objemu přenášených dat při M2M komunikaci znamená použití IP sítí vysokou režii pro koncová zařízení. Velikost prázdného TCP datagramu je minimálně 64 bajtů a jeho použití je vzhledem k většině M2M aplikací neefektivní. Z tohoto důvodu byly vyvinuty specializované protokoly i komunikační sítě určené především pro M2M komunikaci.

1.2.2 Napájení a výpočetní výkon

Specifické požadavky na napájení jsou kladeny především pro koncová zařízení, která díky svému fyzickému umístění nemusí být připojena k elektrické rozvodné síti. Tato zařízení jsou napájena z alternativních zdrojů, především z akumulátorů, které je nutné pravidelně měnit, nebo jsou dobíjeny např. pomocí solárních panelů či miniaturních větrných elektráren [7]. Již nyní však existují zařízení, která výměnu akumulátoru ani dobíjení neumožňují a jejich životnost se odvíjí od výdrže interní baterie [8]. Z těchto důvodů musí mít tato zařízení co možná nejmenší spotřebu elektrické energie. Toho je docíleno použitím energeticky efektivních architektur (SoC – Systém on a Chip) a omezením objemu přenášených dat, snížením četnosti přenosů dat nebo kombinací obojího.

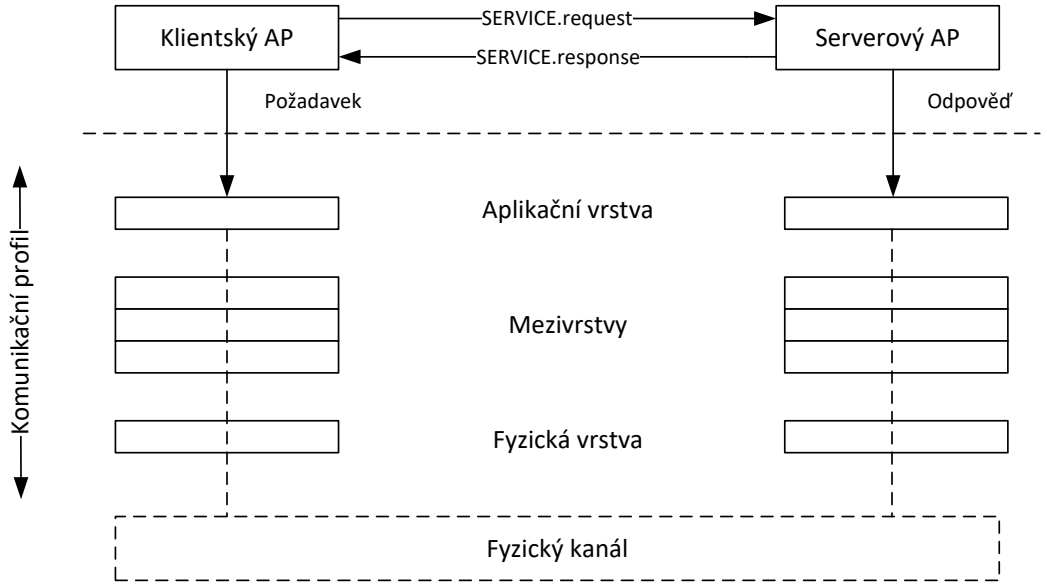
Procesory pro koncová zařízení jsou založeny na architektuře RISC (Reduced Instruction Set Computing). Jejich nižší výpočetní výkon, který je však pro M2M dostatečný, je vyvážen nízkou spotřebou elektrické energie. Průkopníkem ve vývoji těchto procesorů je společnost ARM. V současné době však tyto procesory vyvíjí všichni hlavní výrobci SoC, např. Intel, Qualcomm, atd.

2 DLMS/COSEM

DLMS/COSEM (Device Language Message specification / COmpanion Specification for Energy Metering) je specifikace vytvořená pro odečítání různých údajů z chytrých elektroměrů určující jednotné rozhraní a komunikační protokol. Z pohledu ISO/OSI modelu se DLMS nachází na transportní a relační vrstvě, COSEM poté na vrstvě prezentační.

2.1 Základy komunikace mezi zařízeními

Výměna dat mezi zařízeními dle COSEM protokolu je postavena na modelu klient – server, kde je elektroměr vždy v roli serveru. Dle specifikace neprobíhá výměna dat přímo mezi fyzickými zařízeními, ale mezi aplikačními procesy (AP), které na zařízeních běží. Klientský AP požaduje služby a serverový AP je poskytuje. Požadavek je v prostředí COSEM nazván `SERVICE.request` a odpověď `SERVICE.response`. Jeden serverový AP je schopen obsluhovat souběžně několik klientských a stejně tak klientský AP může komunikovat s více serverovými [9].



Obrázek 1: Klient - server komunikace

Výměna zpráv je podřízena předem určeným komunikačním profilům, které zahrnují několik vrstev (podobně jako u ISO/OSI (International Organization for Standardization / Open Systems Interconnection) modelu, viz Obrázek 1) a mohou být spojově orientované či nikoliv. Aplikační vrstva je vždy řízena protokolem COSEM, nižší vrstvy jsou specifické pro použitý typ komunikačního média. Specifikovány jsou dva základní typy komunikačních profilů [9]:

1. HDLC (High-Level Data Link Control)

   Tento profil je spojově orientovaný a obsahuje 3 vrstvy. Aplikační vrstvu řídí COSEM protokol, spojová vrstva funguje na principu HDLC. Fyzická vrstva

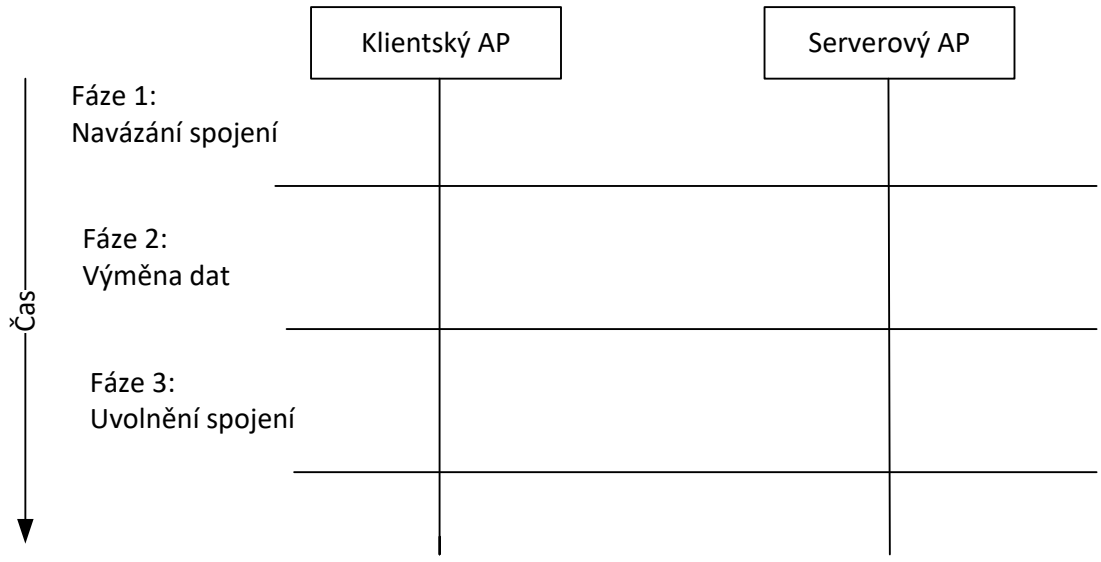
podporuje datovou komunikaci pomocí lokálního optického nebo elektrického portu (RS-485), pronajatých okruhů a telefonních PSTN (Public Switched Telephone Network) nebo GSM (Global System for Mobile Communications) sítí.

2. TCP-UDP/IP

   Jedná se o profil zajišťující výměnu dat přes IP sítě, především Internet, které jsou podporovány řadou fyzických přenosových médií (Ethernet, ISDN (Integrated Services Digital Network), GRPS (General Packet Radio Service)...). Aplikační data COSEM vrstvy jsou přenášena COSEM transportní vrstvou, která se skládá z obalu (wrapper) a TCP nebo UDP protokolu. Nižší vrstvy jsou opět závislé na použitém komunikačním médiu.

DLMS aplikační protokol je spojově orientovaný a pro výměnu dat je tedy nejdříve nutné provést asociaci (sdružení) klienta a serveru. Celá komunikační relace tak sestává ze tří fází (viz Obrázek 2):

- Navázání oboustranného spojení mezi klientem a serverem (asociace)
- Výměna dat (požadavky a odpovědi)
- Uvolnění spojení



Obrázek 2: Schéma komunikační relace DLMS/COSEM

Vzhledem k zaměření této práce jsou další části zaměřeny na komunikační profil využívající HDLC.

2.2 Asociace s elektroměrem

Jak již bylo zmíněno v předchozí části (2.1), před samotnou výměnou dat je nutné klienta se serverem asociovat. Níže popsaný postup se aplikuje výhradně při použití módu „E“ protokolu COSEM. V této situaci se jedná o přímé lokální spojení elektroměru s přenosným zařízením pomocí sériové linky RS-485. Navázání spojení v módu „E“ obsahuje několik částí, během kterých si zařízení vyměňují datové zprávy předem daného formátu a zároveň se příslušně mění parametry sériové linky dle standardu i dle obsahu vyměněných zpráv [9].

2.2.1 Handshake

Prvotní žádost o navázání nového spojení je plně v režii klienta – přenosného zařízení. Server – elektroměr je v této fázi pasivní a čeká na první zprávu od klienta. Pro zajištění kompatibility spojujících se zařízení je rychlost sériové linky nastavena vždy na hodnotu 300 Bd. Další parametry linky musí odpovídat hodnotám v Tabulce 1.

Parametr	**Hodnota**
Rychlost	300 Bd
Počet datových bitů	7
Parita	Sudá
Počet stop bitů	1

Tabulka 1: Parametry sériové linky při navazování spojení

Klient je povinen zaslat serveru pouze jednu zprávu a vyčkat na odpověď. Formát úvodní zprávy od klienta je popsán v Tabulce 2.

Pořadí	**1**	**2**	**3**	**4**	**5**	**6**
ASCII	/	?	A	!	CR (\r)	LF (\n)
HEX	2F	3F		21	0D	0A
DEC	47	63		33	13	10

Tabulka 2: Obsah úvodní zprávy od klienta

Na třetí pozici je vyhrazeno místo pro adresu klientského zařízení, která je odvozena od hardwarové (MAC) adresy. Její uvedení však není povinné a úvodní zpráva se takto zredukuje na velikost 5 bajtů [9].

Pokud je server schopen nové spojení akceptovat, ihned klientovi odpoví zprávou, které obsahuje parametry sériového spojení a identifikací serveru. Struktura zprávy je popsána v Tabulce 3.

Pořadí	1	2	3	4	5	6	7	8	9	10
ASCII	/	X	X	X	Z	\	W	B	CR (\r)	LF (\n)
HEX	2F					5C			0D	0A
DEC	47					92			13	10

Tabulka 3: Odpověď serveru na úvodní zprávu

Tato zpráva je v každém serverovém zařízení nastavena výrobcem a je neměnná. Znaky na pozicích 2, 3 a 4 jsou vyhrazeny pro iniciály výrobce zařízení, kterými však sestavování spojení není ovlivněno. Pro úspěšné navázání spojení je podstatný znak na páté pozici. Obsahuje číslici v rozmezí od 0 do 9, které udává maximální podporovanou rychlost sériové linky ze strany serveru. Při výměně dat budou obě zařízení používat tuto rychlost. Pro její určení je nutné použít převodní tabulku:

Číslice	Rychlost [Bd]
0	300
1	600
2	1200
3	2400
4	4800
5	9600
6	19200
7, 8, 9	rezervovány

Tabulka 4: Převodní tabulka pro rychlost sériové linky

Druhý podstatný údaj se nachází na 7. pozici. Znovu se jedná o číslici, která vyjadřuje komunikační profil pro výměnu dat po úvodní iniciaci spojení. V případě podpory HDLC profilu musí být na této pozici zaslána číslice „2“. Na osmé pozici začíná textový řetězec obsahující celý název výrobce elektroměru.

Poslední zprávou v této úvodní sekvenci je zpráva odeslaná klientem, jejímž obsahem jsou dohodnuté parametry sériové komunikace. Klient tak dává najevo souhlas s použitím přijatých parametrů. Tato zpráva má pevnou délku 5 bajtů a její obsah odpovídá Tabulce 5. Na třetí pozici je zpět serveru zaslána číslice, která představuje finální rychlost sériové linky. Její hodnota je stejná jako v předchozí zprávě zaslané serverem [9].

Pořadí	**1**	**2**	**3**	**4**	**5**	**6**
ASCII	ACK	2	Z	2	CR (\r)	LF (\n)
HEX	2F	32		32	0D	0A
DEC	47	50		50	13	10

Tabulka 5: Potvrzení parametrů sériové linky klientem

2.2.2 Finální potvrzení serverem

V této části server finálně potvrdí navázání nového spojení nebo jej zamítne. Rychlost sériové linky se v této částí mění na hodnotu dohodnutou v předchozím kroku, ostatní parametry však zůstávají shodné s první částí. Jejich přehled je uveden v Tabulce 6.

Parametr	**Hodnota**
Rychlost	Z (dle převodní tabulky)
Počet datových bitů	7
Parita	Sudá
Počet stop bitů	1

Tabulka 6: Nastavení sériové linky při potvrzení serverem

Pokud server spojení potvrzuje, odesílá klientovi zprávu s obsahem shodným s poslední zprávou z předchozího kroku (Tabulka 5). V případě zamítnutí spojení odešle server klientovi zprávu obsahující pouze znak „NAK“. Jeho číselná reprezentace je obsahem Tabulky 7.

Pořadí	**1**
ASCII	NAK
HEX	15
DEC	21

Tabulka 7: Obsah zprávy zamítající spojení

Tímto je veškerá komunikace mezi oběma zařízeními ukončena. Server bude znovu pasivně vyčkávat na úvodní zprávu od klienta [9].

2.2.3 Úspěšné navázání spojení

Pokud server spojení v předchozím kroku přijal, jsou parametry sériové linky upraveny do finální podoby (Tabulka 8) a veškerá následující komunikace se řídí

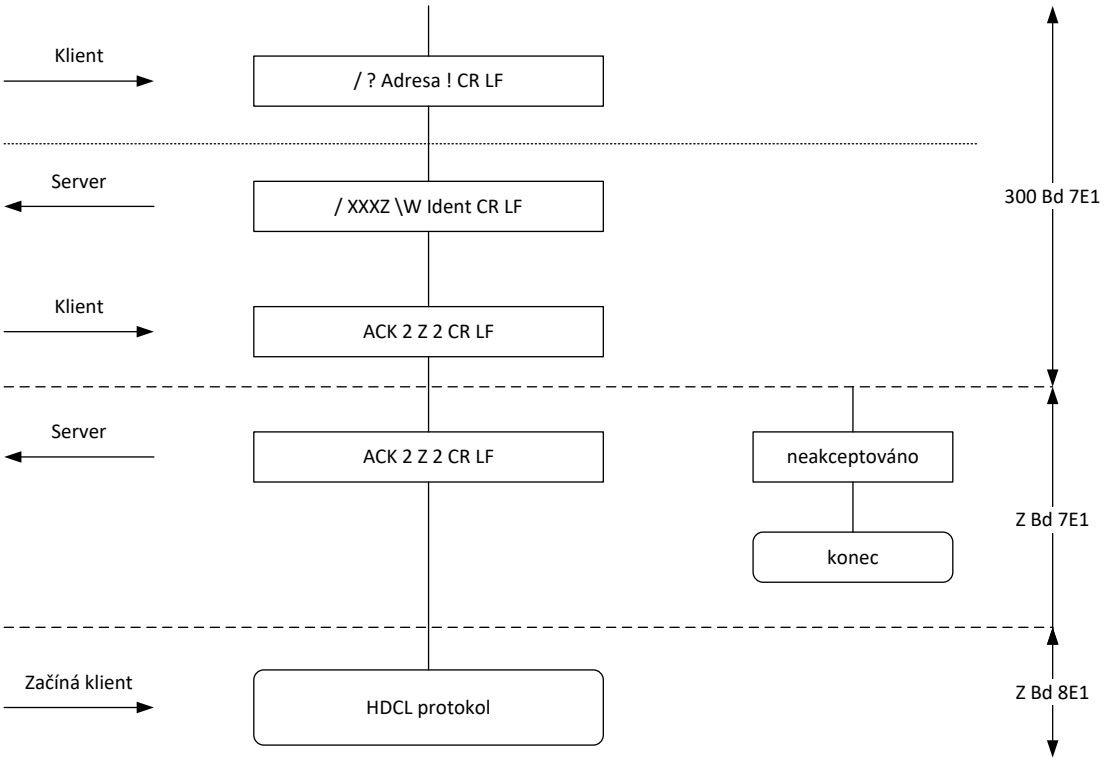
protokolem HDLC (viz kapitola 2.3).

Parametr	Hodnota
Rychlost	Z (dle převodní tabulky)
Počet datových bitů	8
Parita	žádná
Počet stop bitů	1

Tabulka 8: Parametry sériové linky pro komunikaci dle HDLC protokolu

2.2.4 Navázání spojení přes sériovou linku – shrnutí

Spojení s elektroměrem iniciuje odečítací zařízení a celý proces se skládá z celkem čtyř datových zpráv, mezi jejichž odesíláním se mění rychlost sériové linky. Výsledkem procesu je připravené spojení pro HDLC komunikaci nebo odmítnutí a ukončení spojení. Pro názornost je celý proces zobrazen v Obrázku 3 [9].



Obrázek 3: Sestavení sériového spojení

2.3 HDLC komunikace

Po úspěšném navázání spojení mezi serverem a klientem jsou obě zařízení připravena na výměnu dat. Pro tyto účely je na linkové vrstvě využit protokol HDLC, který dovoluje odesílat datové rámce přes synchronní i asynchronní sériové linky. Každý

rámec se skládá z hlavičky, datové části a kontrolního součtu (Obrázek 4). Jednotlivé rámce jsou odděleny pomocí pole nazvaného „Flag field“. DLMS/COSEM využívá HDLC rámce ve formátu typu 3 dle definice v Annex H.4 v ISO/IEC 13239 [9].

Flag	Frame format	Dest. address	Src. address	Control	HCS	Information	FCS	Flag

Obrázek 4: HDLC rámec ve formátu typu 3

2.3.1 Příznak

Jedná se o jeden bajt vkládaný před a za každý HDLC rámec, které tak vzájemně odděluje. Jeho obsah nabývá vždy pouze hodnoty 0x7E (0b01111110). Pokud jsou dva a více rámců odesílány kontinuálně, je mezi nimi příznak (Flag field) vložen pouze jednou (viz Obrázek 5) [9].

Flag	Rámec N	Flag	Rámec N+1	Flag	Rámec N+2	Flag

Obrázek 5: Kontinuální vysílání rámců

2.3.2 Formát rámce

V tomto bitovém poli o celkové délce dva bajty je umístěno označení typu (4 b) a celková délka HDLC rámce (11 b). Obě hodnoty odděluje tzv. „Segmentační bit“. Rozdělení pole na jednotlivé bity je znázorněno na Obrázku 6.

1	0	1	0	S	L	L	L	L	L	L	L	L	L	L	L
Označení typu					Délka rámce										

Obrázek 6: Obsah pole formátu rámce

Část obsahující typ rámce nese vždy hodnotu 10 (0b1010), která identifikuje HDLC rámce typu 3.

Segmentační bit se využívá v případě rozdělení zprávy do více rámců a nabývá těchto hodnot [9]:

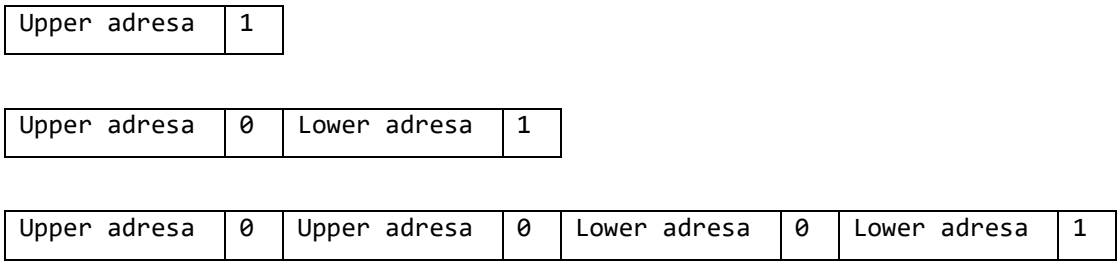
- 0, pokud se jedná o jediný nebo poslední rámec příslušné zprávy
- 1, pokud se jedná o jakýkoliv jiný rámec příslušné zprávy

2.3.3 Adresní pole

Pole obsahuje dvě části pro cílovou a zdrojovou adresu dvou komunikujících zařízení. Délka pole je proměnná v závislosti na použití rozšířeného adresování na straně serveru (extended addressing - ISO/IEC 13239:2002 4.7.1). Toho je využíváno v případě nutnosti adresovat více než jedno logické zařízení v rámci jednoho fyzického. Tato situace může nastat např. v případě zařízení, které je schopné měřit více různých veličin,

ale má jeden společný port pro sériovou linku. Adresa serveru je poté rozdělena na dvě části – upper a lower. Upper adresa se používá pro logická zařízení a musí být vždy přítomna. Lower adresu pro fyzické zařízení je možné vynechat, pokud není požadována.

Pokud server využívá rozšířené adresování, nastaví poslední bit prvního oktetu adresy na hodnotu 0. Adresní pole je možné rekurzivně rozšiřovat nastavením hodnoty 0 do posledního bitu každého dalšího oktetu adresy. Poslední adresní oktet má na poslední pozici hodnotu 1. Takto je tomu i v případě, že se jedná o nerozšířenou jedno-bajtovou adresu [9].



Obrázek 7: Možný formát HDLC adresy serveru

V rámci adresování protokolu HDLC může serverová adresa nabývat délky jeden, dva nebo čtyři bajty. Adresa klienta má vždy délku jeden bajt.

2.3.4 Kontrolní pole

Délka tohoto pole je vždy jeden bajt a obsahuje identifikaci příkazu nebo odpovědi a v případě rámců I, RR a RNR navíc obsahuje sekvenční číslo. Význam jednotlivých typů rámců je uveden v následujícím seznamu [9]:

- I rámec: obsahuje data
- RR rámec: zařízení informuje protistranu, že je připraveno na příjem I rámců a potvrzuje přijetí předchozích I rámců
- RNR: zařízení není dočasně schopné přijímat I rámce např. v důsledku přetížení

Příkaz	Odpověď	Bitový obsah pole
I	I	R R R P/F S S S 0
RR	RR	R R R P/F 0 0 0 1
RNR	RNR	R R R P/F 0 1 0 1

Tabulka 9: Struktura kontrolního pole dle jeho typu

Bity uvnitř pole musí odpovídat struktuře popsané v Tabulce 9 a mají následující význam:

- RRR: Obsahují hodnotu čísla N(R), které slouží jako potvrzení o přijetí

I rámců. Jeho hodnotou je číslo prvního nepřijatého rámce, který by byl přijat jako další v pořadí. Sekvenční číslo posledního přijatého rámce má hodnotu N(R) – 1.

- SSS: Tyto bity obsahují hodnotu čísla N(S), které obsahuje sekvenční číslo odeslaného rámce.
- P/F: jedná se o tzv. Poll/Final bit. Název Poll využívá stanice v případě odeslání příkazu, na který očekává odpověď. Naopak jako Final se bit nazývá, pokud je odeslána odpověď na příkaz nebo se jedná o konec vysílání.

2.3.5 Kontrolní součet hlavčiky (HCS)

HCS (Header Check Sequence) je pole o délce dvou bajtů. Musí být obsažen ve všech HDLC rámcích bez prázdného informačního (datového) pole [9]. Kontrolní součet je poté vypočítán ze všech ostatních polí hlavičky rámce (Formát rámce, Adresní pole, Kontrolní pole). Výpočet hodnoty kontrolního součtu probíhá pomocí předem definované vyhledávací tabulky (lookup table), počáteční hodnoty (0xFFFF) a výpočetní funkce, jejíž zdrojový kód je součástí přílohy této práce.

2.3.6 Informační pole

Do tohoto pole jsou umístěna data a může se tak jednat o libovolnou sekvenci bajtů. V případě I rámců je jeho obsahem MSDU (MAC Service Data Unit).

2.3.7 Kontrolní součet rámce

Principiálně je toto pole shodné s kontrolním součtem hlavičky (viz sekce 2.3.5). Má shodnou délku 2 bajty a pro jeho výpočet je použita stejná funkce. Hodnota součtu je vypočítána ze všech polí v rámci, tedy z hlavičky, jejího HCS a informačního pole [9].

2.3.8 Transparentnost

Oddělování rámců pomocí Flag field vnáší do mechanismu odesílání dat další problematiku. Pokud by data uvnitř rámce obsahovala hodnotu 0x7E, příjemce zprávy by tento bajt mylně považoval za ukončení rámce a jeho zpracování a následné zpracování zbytku zprávy by vedlo k chybě v komunikaci. Pro odeslání rezervovaných hodnot je v HDLC rámcích použita technika „octet stuffing“, při níž je jeden bajt nahrazen dvěma. První z těchto bajtů se nazývá „Control Escape octet“ a nabývá hodnoty 0x7D. Hodnota druhého bajtu je vypočítána z jeho původní hodnoty pomocí logické operace XOR s hodnotou 0x20 (0b10100). Využití „Control Escape“ oktetu vyžaduje, aby jeho hodnota nacházející se uvnitř datového rámce byla rovněž ošetřena technikou „octet stuffing“.

PŮVODNÍ HODNOTA	ZAKÓDOVANÁ HODNOTA	VÝZNAM
0x7E	0x7D, 0x5E	Flag field
0x7D	0x7D, 0x5D	Control Escape

Tabulka 10: Kódování rezervovaných hodnot

Zakódování rezervovaných hodnot se provádí až po výpočtu kontrolního součtu (FCS – Frame Check Sequence). Příjemce poté nejdříve hodnoty uvozené bajtem „Control Escape“ dekóduje a až poté vypočítá kontrolní součet. Dekódovací proces nejdříve odstraní všechny bajty o hodnotě 0x7D a původní hodnotu získá pomocí funkce XOR z hodnoty následujícího bajtu a hodnoty 0x20. Převod rezervovaných hodnot je zobrazen v Tabulce 10 [9].

2.3.9 MSDU

MAC Služební datová jednotka (Service Data Unit) se nachází uvnitř informačního pole HDLC rámce a obsahuje svou vlastní hlavičku a aplikační data. Struktura datové jednotky je vyobrazena na Obrázku 8 [9].

Destination (8b)	Source (8b)	LLC_Quality (8b)	Data (n × 8b)

Obrázek 8: Struktura MSDU v prostředí DLMS/COSEM

- Destination: obsahuje vždy hodnotu 0xE6
- Source: nabývá pouze dvou hodnot – 0xE6 v případě odeslání příkazu, 0xE7 v případě odpovědi
- LLC_Quality: toto pole je rezervováno pro budoucí použití a nyní musí být vždy nastaveno na hodnotu 0x00
- Data: Datová část o délce N bajtů ($0 \leq N$)

2.4 OBIS (Object Identification Systém) kód

Součástí protokolu COSEM je systém pro adresaci objektů, tzv. OBIS. Jedná se o soubor 6 bajtových identifikačních kódů, které slouží jako logické názvy pro jednotlivé objekty uchovávané v paměti měřícího zařízení. Takto adresované objekty obsahují naměřené údaje, které je možné odečíst. Pro odečtení konkrétního údaje je tedy nutné předem znát jeho OBIS kód (viz Tabulka 11).

Seznam všech OBIS kódů s jejich popisem (tj. významem údaje, který se v daném objektu uložen) je spravován a vydáván organizací DLMS User Association jako standard. Jednotlivé kódy jsou nezávislé na výrobci měřícího zařízení, ale jsou pro všechny výrobce závazné, čímž je zajištěna univerzálnost aplikací pracujících s protokolem COSEM [24].

1.0.1.0.8.255

Tabulka 11: Příklad OBIS kódu

3 PRAKTICKÁ ČÁST

Primárním úkolem této práce bylo zpracovat teoretické údaje o DLMS/COSEM protokolu a získané poznatky využít při vývoji aplikace pro OS Android schopné provádět následující operace:

- Připojit se k elektroměru pomocí USB OTG (On The Go).
- Navázat oboustrannou komunikaci.
- Získat identifikaci elektroměru.
- Uložit naměřené hodnoty do vzdálené databáze

V této kapitole jsou detailně popsány dílčí kroky, které bylo nutné vyřešit při programování mobilní aplikace a jejím testování.

3.1 Hardwarové vybavení

3.1.1 Přenosné měřící zařízení

I když je možné provozovat operační systém Android na klasickém počítači postaveném na architektuře x86 (Remix OS) [10], pro testování aplikace bylo zvoleno prostředí, které více odpovídá praktickému využití této aplikace. Z tohoto důvodu bylo zvoleno zařízení Google Nexus 7 druhé generace. Tento tablet vyrobila společnost Asus a na trh byl uveden v září roku 2013. I přes stáří 3 roky nabízí více než dostatečnou hardwarovou výbavu pro testování vyvíjené aplikace (viz Tabulka 12). Původně dodávaný operační systém Android 4.3 (Jelly Bean) byl před vývojem aplikace aktualizován na verzi 5.1 (Lollipop). Žádné další modifikace nebyly na tabletu provedeny.

Výbava	**Hodnota**
Procesor	1,5 GHz, 4 jádra
RAM	2GB
Displej	7“, 1200 × 1920 px
Interní úložiště	16GB
OS	Android 5.1 (Lollipop)
USB OTG	ANO (micro USB 2.0)
Baterie	3950 mAh

Tabulka 12: Výbava testovacího tabletu Nexus 7

Z výše uvedených parametrů je pro vyvíjenou aplikaci nejpodstatnější podpora USB

OTG a baterie s dostatečnou kapacitou pro napájení samotného tabletu i zařízení připojeného přes USB OTG.

3.1.2 Optický převodník pro sériovou linku

Vybraný tablet musel podporovat funkci USB OTG, která tvoří základ pro praktické využití vyvíjené aplikace. Přes toto rozhraní je realizováno spojení mezi tabletem a elektroměrem a probíhá přes něj komunikace dle standardu DLMS/COSEM. Jako fyzické médium pro přenos dat slouží sériová linka (RS-485), kterou však tablet nativně nepodporuje. Proto byl použit převodník mezi rozhraními USB a RS-485. Pro připojení do USB portu slouží kabel s koncovkou USB Typ-A. Sériové rozhraní převodníku je provedeno v optické variantě. Toto řešení umožňuje propojit elektroměr s tabletem pouhým zachycením magnetického optického převodníku ve svorce na tělo elektroměru. Protože jsou USB konektory tabletu a převodníku vzájemně nekompatibilní, byl pro jejich propojení použit USB OTG kabel sloužící jako redukce z USB na micro USB konektor. Převodník je poté v operačním systému identifikován jako nativně podporovaná sériová linka a není pro něj tedy nutné instalovat ovladače nebo jiný software.

3.1.3 Elektroměr Kaifa MA110

Jedná se o jednofázový elektroměr určený pro elektrické rozvodné sítě s nominálním napětím 230 V. Jakožto přístroj určený i do chytrých domácností je vybaven mechanismem pro vzdálené odečítání dat. K tomuto účelu slouží několik rozhraní:

- optický port pro DLMS/COSEM (mód E)
- RS485 port
- bezdrátová sběrnice M-Bus
- GPRS modul pro IP DLMS/COSEM

Pro komunikaci s mobilním zařízením je využit pouze optický port [11].

Elektroměr byl vyroben společností Shenzhen Kaifa Technology Co., Ltd se sídlem v Číně. Jeho zapůjčení bylo zprostředkováno rakouskou společností Telekom Austria AG.

Zapojení elektroměru je zobrazeno na fotografii 1 v příloze.

3.2 Databáze

3.2.1 Databázový software

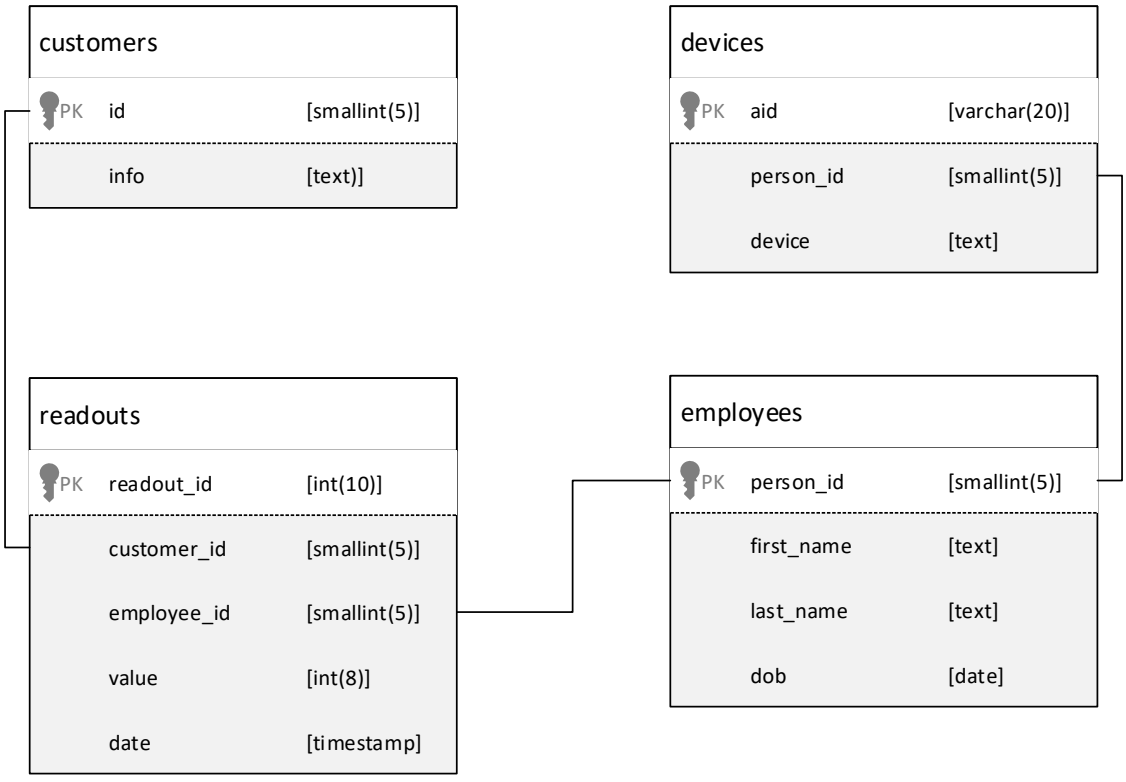
Pro správné fungování vyvíjeného řešení je nutné ukládat získaná data do centrální databáze. Pro tento účel byl vybrán databázový systém MariaDB. Jedná se o odnož databázového systému MySQL, se kterým je MariaDB zpětně kompatibilní. K oddělení bylo přistoupeno v roce 2009 po akvizici MySQL společností Oracle. Záměrem oddělení je snaha původních vývojářů MySQL udržet projekt na trhu jako otevřený software dostupný zdarma. Oproti MySQL je přidána podpora více metod pro ukládání dat (storage engines), podstatně byla vylepšena i rychlost zpracování dotazů a především vnořených

dotazů (subqueries) [14]. Celosvětově je MariaDB jeden z nejpopulárnějších databázových systémů, což dokazuje i jeho používání ve velkých internetových společnostech (Google, Wikipedia, ...). Jeho blízkost k linuxovým operačním systémům, kterým je i Android, z něj současně utváří ideální volbu pro použití v praktické části této práce.

Pro účely této práce byla MariaDB databáze (verze 10.0.26) zprovozněna na dedikovaném serveru s neomezenou internetovou konektivitou o rychlosti 1 Gbps. Server je provozován pod operačním systémem Debian ve stabilní verzi 8.5 (jessie).

3.2.2 Databázové schéma, význam tabulek a atributů

V této části je popsána logická struktura relační databáze, která byla navrhnuta s ohledem na možné potřeby společnosti využívající tento systém sběru dat. V databázi tedy byly připraveny tabulky pro uložení naměřených hodnot a identifikaci zaměstnanců, jejich mobilních zařízení a jednotlivých zákazníků. Výhodou použití relační je možnost jejího snadného rozšíření o další tabulky v případě potřeby. Schéma použité databáze je zobrazeno na Obrázku 9.



Obrázek 9: Schéma použité databáze

- employees

Tato tabulka slouží pro uchovávání dat o zaměstnancích. Pro zjednodušení jsou obsaženy pouze základní osobní informace. Seznam atributů s popisem je uveden v Tabulce 13Tabulka 13.

Atribut	Typ	Parametry	Význam
person_id	smallint(5)	UNSIGNED,NOT_NULL, AUTO_INCREMENT	Jednoznačné označení zaměstnance
first_name	text	NOT_NULL, UTF-8	Křestní jméno
last_name	text	NOT_NULL, UTF-8	Příjmení
dob	date	NOT_NULL	Datum narození

Tabulka 13: Význam atributů tabulky "employees"

- readouts

V této tabulce budou uloženy všechny odečtené údaje o spotřebě. Kromě samotné odečtené hodnoty je nutné uchovávat i údaje, které měření provázely jako datum a čas, identifikace zaměstnance a zákazníka (viz Tabulka 14).

Atribut	Typ	Parametry	Význam
readout_id	int(10)	UNSIGNED, NOT_NULL, AUTO_INCREMENT	Automaticky se navyšující ID. Slouží pro usnadnění případné manipulace s daty v této tabulce
customer_id	smallint(5)	UNSIGNED, NOT_NULL	Jednoznačné označení zákazníka, u kterého byl odečet proveden
employee_id	smallint(5)	UNSIGNED, NOT_NULL	Jednoznačné označení zaměstnance, který odečet provedl
value	int(8)	UNSIGNED, NOT_NULL	Odečtená hodnota o spotřebě
date	timestamp	NOT_NULL, CURRENT_TIMESTAMP	Automaticky doplněné datum a čas odečtu

Tabulka 14: Význam atributů tabulky "readouts"

- devices

Třetí tabulka obsahuje údaje o mobilních zařízeních, pomocí kterých se provádí odečítání údajů z elektroměrů. Pro jednoznačnou identifikaci přístroje je využit kód Android ID, který je daným zařízením vygenerován během prvního spuštění. Toto ID je možné změnit pouze uvedením zařízení do továrního nastavení a jeho další výhodou je nezávislost na hardwarové výbavě daného zařízení, čímž jsou eliminovány problémy s hardwarovými identifikátory. Pokud by byl pro identifikaci použit např. kód IMEI

(International Mobile Equipment Identity) a zařízení nedisponovalo slotem pro SIM kartu, nebylo by možné toto zařízení identifikovat a použít. Tímto je zároveň vyřešena jednoznačná identifikace zaměstnance, který odečet provedl, a ukládání dat pouze z autorizovaných zařízení, protože Android ID přístroje musí být v databázi uloženo předem. Význam atributů tabulky „devices“ je popsán v Tabulce 15.

Atribut	**Typ**	**Parametry**	**Význam**
person_id	smallint(5)	UNSIGNED,NOT_NULL, AUTO_INCREMENT	ID zaměstnance, kterému přístroj patří
aid	varchar(20)	NOT_NULL	Android ID zařízení
device	text	NULL	Textový popis zařízení, např. výrobce a model

Tabulka 15: Význam atributů tabulky "devices"

- customers

Poslední databázová tabulka je určena pro uchovávání údajů o zákaznících. Obsahuje pouze dva sloupce – ID a „info“. Protože cílem práce nebylo vytvoření databáze zákazníků, do pole „info“ je možné vložit libovolný popis daného zákazníka. V rámci práce je uloženo celé jméno a město. Parametry atributů jsou popsány v Tabulce 16.

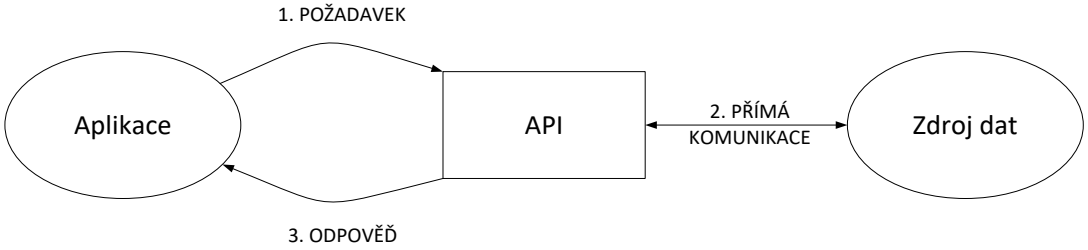
Atribut	**Typ**	**Parametry**	**Význam**
id	smallint(5)	UNSIGNED,NOT_NULL, AUTO_INCREMENT	ID zákazníka
info	text	NOT_NULL	Jméno a město zákazníka

Tabulka 16: Význam atributů tabulky "customers"

3.3 SQL API

3.3.1 Komunikace mezi aplikací a MariaDB

V době psaní této práce nebyla pro OS Android dostupná oficiální knihovna pro přímou komunikaci s MySQL serverem. Bylo možné použít knihovny třetích stran, avšak řešení využívající přímé komunikace s DB není vzhledem k použití mobilního zařízení a s ohledem na omezené systémové zdroje vhodné. Pro přenos dat mezi aplikací a DB bylo tedy žádoucí využít prostředníka, tzv. API (Application Program Interface). Všeobecně se jedná se o software, který slouží jako mezivrstva pro komunikaci mezi dvěma aplikacemi (Obrázek 10). API nabízí unifikovanou metodu přístupu k jednomu zdroji dat pro různé aplikace. K této komunikaci API využívá předem daný komunikační protokol, který navrhuje tvůrce API a je uzpůsoben konkrétnímu případu použití.



Obrázek 10: Schéma komunikace s využitím API

Ze schématu je zřejmé, že získání zdrojových dat je plně v režii API, což vede k úspoře dat při komunikaci s databází. Aplikace pouze vysílá požadavek k API a následně obsluhuje přijatá data. Tato data navíc mohou být na úrovni API předpřipravena do strukturovaného formátu (např. XML - Extensible Markup Language) za účelem dalšího snížení režie při zpracování dat aplikací. To napomáhá k šetření systémových zdrojů zařízení, na kterém je aplikace spuštěna.

K další úspoře dochází na komunikační vrstvě, převážně pokud se jedná o mobilní datovou síť, kde může být komunikace zpoplatněna na základě objemu přijatých a odeslaných dat. Požadavek na API je úmyslně formulován jako co nejkratší možný řetězec obsahující instrukci a případně parametry. Instrukce API přikazuje, kterou operaci má s daty zdroje provést. Různé instrukce mohou být doplněny různými parametry potřebnými k provedení požadované operace. Kontrola validnosti přijatých požadavků je opět v režii API. Po validaci následuje zpracování a odeslání dat zpět aplikaci. V případě neúspěšné validace je zpět odesláno upozornění s odůvodněním, proč ke zpracování dat nedošlo. Aplikace tedy musí být schopna zpracovat nejen přijatá data, ale i přijatá oznámení vygenerovaná API. Součástí validace požadavku na straně API mohou být i systémy autentizace a autorizace. Pro mobilní zařízení se tedy ve všech ohledech jedná o velmi vhodné řešení.

3.3.2 Vývoj API

Před samotným procesem programování API bylo nutné rozhodnout, na jaké technologii bude postaveno. Žádoucí bylo vyvinout multiplatformní software, protože k API mohou přistupovat různé systémy a aplikace, které jsou navzájem nekompatibilní. Z tohoto důvodu bylo zvoleno API v podobě webové aplikace. K provozu tohoto typu API je jedinou prerekvizitou provozování webového serveru, který je však možné spustit na všech běžných typech operačních systémů. V případě této práce bylo vhodné webové API použít z toho důvodu, že webový server je spuštěn na stejném fyzickém serveru, na kterém je nainstalována databáze MariaDB, ke které bude API přistupovat.

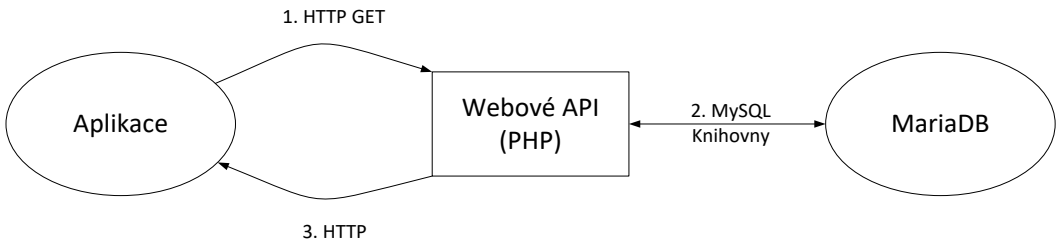
Díky použití webového serveru odpadá nutnost zabývat se vývojem komunikační metody mezi aplikací a API. Komunikace mezi aplikací a API bude probíhat prostřednictvím HTTP (Hypertext Transfer Protocol), což je nejrozšířenější protokol pro internetovou komunikaci. Jeho podpora je zajištěna ve všech moderních operačních systémech včetně OS Android a je tak zajištěna i kompatibilita pro budoucí vyvíjené verze. Další výhodou použití webového API byla možnost ladění samotného programu pomocí webového prohlížeče. Vývojář tedy nemusí mít k dispozici software, který bude s API komunikovat. Vzhledem k operačnímu systému Debian byl zvolen Apache (verze

2.4) jako software pro webový server. Apache je v základním nastavení schopný obsloužit až 256 současně připojených klientů, není tedy nutné se zabývat řešením problémů souvisejících se souběžným přístupem požadavků k API [15].

Dalším krokem při vývoji API byl výběr programovacího jazyka, ve kterém bude API napsáno. Vzhledem k použití webového serveru Apache bylo nutné vybírat z jazyků, které Apache podporuje. Nejpopulárnějšími podporovanými jazyky jsou unixové skriptovací jazyky Python a Perl nebo PHP, které bylo primárně vyvinuto pro webové aplikace. Doinstalovat podporu dalších jazyků je možné v podobě modulů pro Apache. Webové API, které je součástí této práce, bylo naprogramováno v jazyce PHP (verze 5.6). Jde o interpretovaný programovací jazyk, který je jako modul součástí softwaru Apache. Pro programování v tomto jazyce nejsou potřeba žádné specializované vývojové nástroje, zdrojový kód se zadává přímo do těla webové stránky jako prostý text. PHP bylo také vybráno z důvodu autorových dřívějších zkušeností s tímto jazykem.

3.3.3 Komunikační protokol

Posledním krokem před přistoupením k samotnému programování bylo vytvoření komunikačního protokolu mezi aplikací a API. Protokol stanovuje přesně daný formát požadavků na API a přesný formát odpovědí na tyto požadavky. Z důvodu využití webového API bylo nutné brát ohled na možnosti protokolu HTTP, který je ke komunikaci použit. HTTP používá k určení zdroje dat řetězec URL, který je zadán formou prostého textu. Je v něm obsažena adresa serveru a cesta k požadovanému souboru. K zasílání požadavků na server využívá HTTP několika metod, z nichž nejpodstatnější jsou GET a POST. Metoda POST je využívána především pro odesílání dat z webových formulářů a je považována za více bezpečnou. Data odeslaná touto metodou jsou zakódována v těle požadavku, nejsou tedy pro uživatele viditelná a nikdy se neukládají do mezipaměti. Při použití metody GET jsou odesílaná data pro uživatele viditelná, protože jsou součásti URL adresy. Díky tomu není nutné data zakódovat tak, jak je tomu u metody POST a odesílání takových požadavků je režijně jednodušší. Pro odesílání požadavků k API je použita metoda GET, která umožňuje vytváření požadavků na úrovni spojování textových řetězců. I když se jedná o zobrazitelnou formu dat, před uživatelem budou odesílané požadavky skryté díky zakomponované podpoře HTTP přímo v OS Android. Schéma finální struktury komunikace mezi aplikací a API je zobrazeno na Obrázku 11.



Obrázek 11: Komunikační schéma navrhovaného API

Po určení komunikační metody bylo nutné navrhnout samotný protokol. Dle zadání práce bylo nutné zapisovat data do databáze. API by tedy mohlo být naprogramováno pouze pro tento případ použití. Pro možnost budoucího rozšíření však bylo API

naprogramováno k rozpoznání dvou příkazů, které lze v případě potřeby snadno rozšířit o další. Prozatím jsou podporovány příkazy pro zápis dat do databáze a získání údajů o zákazníkovi z databáze. V Tabulce 17 je uveden popis příkazů a jejich parametrů.

Příkaz (q)	**Parametr**	**Popis**
w **(write)** zápis do DB	cid	Identifikační kód zákazníka
	aid	Identifikační kód mobilního zařízení
	val	Zapisovaná hodnota
r **(read)** informace o zákazníkovi	cid	Identifikační kód zákazníka

Tabulka 17: Příkazy pro API

Příklad URL pro zápis hodnoty do DB:

```
http://navtom.cz/dp/api.php?q=w&cid=123&aid=112233445566778&val=9019
```

Tímto způsobem byl zajištěn jednotný přístup k datům v databázi. Může k ní přistupovat jakákoliv aplikace podporující HTTP a navrhnutý komunikační protokol. Pro správné fungování API bylo dále nutné navrhnout formát dat, která bude API odesílat zpět aplikaci. Jak již bylo zmíněno, aplikace má data do databáze pouze zapisovat. API by tedy na první pohled nemělo být naprogramováno k odesílání jakýchkoliv dat zpět aplikaci. Je však nutné ošetřit chybové stavy, které mohou při práci s API nastat a aplikaci o nich informovat. V případě zápisu se jedná především o neúspěšnou validaci dat zaslaných aplikací a o neúspěšném zápisu dat do DB. Pro tyto případy by bylo postačující, kdyby API vracelo pouze obyčejný text s kódem chyby. V rámci přípravy na budoucí rozšíření aplikace bylo však vhodné navrhnout strukturovaný formát vracených dat již v této fázi projektu.

Pro strukturovaný výpis textových dat je vhodný formát XML. Jedná se o značkovací jazyk, na jehož principu je postaveno např. HTML (HyperText Markup Language) pro tvorbu webových stránek. Má pevně daná pravidla pro strukturu a je tedy snadné jej strojově číst – parsovat. Dále je možné strukturu samotného dokumentu přizpůsobit dle potřeb vývojáře. Z toho vyplývá, že pro použití v tomto projektu bylo nutné navrhnout strukturu XML dokumentu, který bude výstupem z API. XML dokument má povinné pouze dvě části – hlavičku a tělo. Hlavička uvozuje XML dokument a tělo tvoří dvě párové značky. I když je tělo prázdné, jedná se o validní XML dokument. Základní kostra generovaného XML dokumentu vypadá následovně:

```
<?xml version='1.0' encoding='UTF-8'?>
<body>
</body>
```

Uvedený kód představuje základ při tvorbě výstupního XML dokumentu API. Tělo

uvozené značkou `<body>` bude obsahovat návratové kódy API a v případě budoucího rozšíření data získaná z databáze.

Návratové kódy je nutné navrhnout s ohledem na jejich jednoduchost, avšak chybový kód by měl být doplněn detailním chybovým výpisem pro vývojáře aplikace. Z důvodu jednoduchosti první verze API budou použity pouze tři návratové kódy vypsané v Tabulce 18.

Číselný kód	**Text**	**Popis**
0	DONE	Operace byla úspěšné dokončena
1	DBERR	Chyba při komunikaci s MariaDB
99	QUERY	Chybně zadaný příkaz a/nebo parametry

Tabulka 18: Návratové kódy API

Chybový kód „1 DBERR“ bude vždy, pokud to situace umožní, doplněn o výpis chybové zprávy z MySQL knihovny.

3.3.4 Programování API

V této části jsou uvedeny podstatné části zdrojového kódu PHP API, které jsou pro správné fungování aplikace klíčové. Kompletní zdrojové kódy jsou dostupné v příloze práce. I když je PHP objektově orientovaný programovací jazyk, je API naprogramováno procedurálně. Tato metoda byla vybrána z důvodu celkové jednoduchosti API.

- Typ dokumentu:

Protože je PHP primárně používáno pro dynamické generování webových stránek, je ve všech PHP skriptech automaticky používán typ dokumentu „text/html“ [16]. Díky tomuto údaji webový prohlížeč pozná, že má příchozí data zobrazit jako webovou stránku. Tento údaj odesílá webový server v HTTP hlavičce a PHP jej umožňuje měnit dle potřeb programátora. Pro operace s HTTP hlavičkou slouží v PHP funkce `header()`. Parametrem této funkce je textový řetězec obsahující název položky HTTP hlavičky a její požadovanou hodnotu. Pro správné fungování API bylo nutné nastavit typ dokumentu na „text/xml“. Toho bylo docíleno následujícím použitím funkce `header()`:

```
header('Content-Type: text/xml');
```

Při použití této funkce je nutné brát na vědomí, že jelikož upravuje HTTP hlavičku, je nutné ji v PHP skriptu umístit před kterýkoliv příkaz vypisující data do výsledného dokumentu. Uvození PHP skriptu značkou `<?php` musí být umístěno na první pozici prvního řádku. Jakékoliv odsazení nebo odřádkování by způsobilo odeslání těla dokumentu a úprava hlavičky by již nebyla možná – použití funkce `header()` by mělo za následek neúspěšné provedení celého skriptu.

- SQL příkaz pro vložení dat:

Jak již bylo zmíněno v části zabývající se komunikačním protokolem API, data určená k zápisu do databáze jsou skriptu zasílána v podobě HTTP GET dotazu. Z těchto dat je nutné sestavit validní SQL příkaz, který data do databáze uloží. SQL příkaz je ve své podstatě textovým řetězcem a pro jeho sestavení byla použita funkce `sprintf()`:

```
$sql = sprintf("INSERT INTO `readouts` (`readout_id`, `customer_id`,
`employee_id`, `value`, `date`) VALUES (NULL, '%u', (SELECT `person_id`
FROM `devices` WHERE `aid` = '%u'), '%u', CURRENT_TIMESTAMP);",
$_GET['cid'], $_GET['aid'], $_GET['val']);
```

Jediným povinným parametrem této funkce je textový řetězec. Tímto použitím je však vrácen zadaný textový řetězec přesně tak, jak byl zadán. Funkce se tedy používá pro sestavení textového řetězce pomocí dat uložených v různých proměnných, které mohou být různého typu. Na požadované místo ve vstupním řetězci je umístěn znak %, který následuje jedno-písmenné označení datového typu proměnné, jejíž obsah má být na požadované místo vložen. Za datovým typem může být nepovinně zadán formát dat, např. počet desetinných míst, která mají být zobrazena u datového typu float. Proměnné s vkládanými daty jsou funkci předány jako další parametry. Data z více proměnných jsou do řetězce vkládána na základě pořadí, ve kterém jsou funkci proměnné předány. Pokud je pro vložení proměnné do řetězce použit chybný datový typ, pokusí se PHP o přetypování proměnné. Pokud není přetypování možné, je skript ukončen s chybou [17].

Samotný SQL příkaz se skládá z klíčových slov „INSERT INTO“, které uvozují vkládání dat do DB. Následuje jméno tabulky, do které se mají data vložit – „readouts“. Zpětné apostrofy jsou použity jako ochrana před SQL Injection [18], tedy před formou databázových útoků. Za jménem tabulky následuje nepovinná část, která udává názvy a pořadí sloupců tabulky, do kterých budou data vložena. Tuto část je vhodné zadat jako prevenci před programátorskými chybami. Bez zadaných názvů sloupců se uvažuje pořadí tak, jak bylo zadáno při vytváření databázové tabulky. Klíčové slovo „VALUES“ označuje část, která obsahuje vkládaná data. Pořadí vkládaných hodnot musí odpovídat pořadí názvů sloupců z předchozí nepovinné části. Pro lepší přehled jsou vkládané hodnoty uvedeny v Tabulce 19.

Název sloupce v DB	**Vkládaná hodnota**	**Popis**
readout_id	NULL	Automaticky doplní DB (AUTO_INCREMENT)
customer_id	$_GET['cid']	Z mobilního zařízení
employee_id	Vnořený SQL dotaz	
value	$_GET['val']	Z mobilního zařízení
date	CURRENT_TIMESTAMP	Automaticky doplní DB

Tabulka 19: Rozpis SQL příkazu pro vložení dat

Pro vložení ID zaměstnance, který provedl odečet údajů z elektroměru je použit vnořený SQL dotaz:

```
SELECT `person_id` FROM `devices` WHERE `aid` = '%u';
```

Identifikaci pracovníka určuje Android ID použitého mobilního zařízení a tato data jsou uložena v další databázové tabulce. Tento dotaz vrací ID zaměstnance na základě Android ID, které odesílá mobilní zařízení jako součást HTTP GET požadavku. Tímto způsobem je na úrovní databáze zajištěno vložení dat pouze od autorizovaného mobilního zařízení. Pokud Android ID v tabulce „devices“ nebude nalezeno, databáze vrátí jako výsledek tohoto dotazu tzv. prázdnou hodnotu (NULL). Tuto hodnotu se následně pokusí vložit do sloupce „employee_id“. Při návrhu tabulky „readouts“ byl však tomuto sloupci zadán parametr, který zakazuje vkládání prázdných hodnot (NOT_NULL parametr). Při zápisu hodnoty NULL tedy automaticky dojde k chybě a žádná data nebudou do databáze vložena.

3.3.5 Shrnutí API

Výhody použití webového API pro komunikaci s databází MariaDB je možné shrnout do následujících bodů:

- Vyhnutí se používání neoficiální knihovny pro komunikaci s databází
- Využití zabudovaných schopností systému OS Android (HTTP)
- Šetření systémových prostředků mobilního zařízení
- API je univerzální a použitelné dalšími aplikacemi

Nevýhody:

- Vedle databázového serveru je nutné provozovat i server webový
- Programátor musí ovládat jazyky podporované webovým serverem

3.4 Aplikační data

Díky standardu protokolu DLMS/COSEM byla před vývojem aplikace známá struktura jednotlivých rámců, pomocí kterých elektroměr komunikuje s připojeným zařízením přes sériovou linku. Dále byl znám význam jednotlivých polí v hlavičkách HDLC rámce a MSDU (např. adresní pole, kontrolní součet atd.). Ze standardu však nebylo možné zjistit jakákoliv aplikační data, tedy ta, která jsou přenášena v datovém poli MSDU a obsahují např. naměřené hodnoty spotřeby elektrické energie.

3.4.1 Získání aplikačních dat

Protože výrobce elektroměru Kaifa tyto údaje veřejně neposkytuje, bylo je nutné získat jinou metodou. Pro účely této práce byla aplikační data získána odposlechem komunikace mezi PC a elektroměrem. Ke komunikaci s elektroměrem byl na PC s OS Windows zprovozněn software SmartSet, který je společností Kaifa dodáván k testování jejich elektroměrů – je vybaven sadou příkazů pro odečet různých údajů z elektroměru. Záznam komunikace probíhal pomocí softwaru „HHD Software Device Monitoring Studio“ [25], který veškerou komunikaci probíhající přes zvolené USB rozhraní ukládá pro pozdější analýzu. Výhodou použití tohoto softwaru je jeho schopnost zaznamenávat stav sériové linky, tedy např. během připojování k elektroměru jsou kromě přenášených

rámců zaznamenány parametry sériové linky, z nichž nejdůležitější je rychlost linky uváděná v Bd. Zachytávání komunikace se odehrávalo podle následujícího scénáře:

1. Připojení USB optického senzoru do portu na PC.
2. Odstartování záznamu komunikace.
3. Navázání spojení s elektroměrem přes software SmartSet.
4. Odeslání požadavku na přístup k vybrané hodnotě.
5. Zapsání získané hodnoty přes software SmartSet.
6. Odpojení od elektroměru přes software SmartSet.
7. Zastavení zaznamenávání komunikace.

Touto metodou byla zaznamenána aplikační data pro následující operace:

1. Připojení k elektroměru.
2. Odečtení údaje o celkové naměřené spotřebě (tzv. A-plus záznam).
3. Odpojení od elektroměru.

Záznam komunikace byl několikrát zopakován, aby byl k pozdější analýze k dispozici dostatek dat. Zároveň bylo nutné odečíst různé hodnoty celkové spotřeby, aby bylo v aplikačních datech možné zjistit pozici, na které je tento údaj přenášen, a případně pozici dalších proměnlivých údajů.

3.4.2 Analýza aplikačních dat

Po uložení zaznamenaných dat byla dalším krokem jejich analýza, aby bylo možné zjistit význam jednotlivých přenášených bajtů. Výstupem z programu „HHD Software Device Monitoring Studio“ byl dokument rozdělený na bloky, které reprezentovaly jednotlivé přenášené zprávy a aktuální parametry sériové linky během zasílání těchto zpráv. Dále byly jednotlivé zprávy rozlišeny podle jejich odesílatele. Díky tomuto počátečnímu rozlišení jednotlivých zpráv bylo možné k jejich analýze přistoupit ihned. Analýza probíhala v tabulkových procesorech Microsoft Excel a LibreOffice Calc. Zprávy byly přepsány do oddělených bloků, kdy jedna buňka odpovídala jednomu přenesenému bajtu. Bloky byly opět barevně odlišeny podle odesílatele zprávy v daném bloku. Tímto způsobem bylo vedle sebe vyobrazeno několik zaznamenaných komunikací mezi elektroměrem a PC.

Následně byly porovnány hodnoty bajtů na stejné pozici napříč zaznamenanými komunikacemi s cílem identifikovat bloky, které jsou shodné. Z tohoto kroku vyplynulo zjištění, že pouze dva bloky o délkách 16 a 12 bajtů obsahují proměnlivé hodnoty. Zbytek zpráv byl shodný ve všech uložených komunikacích. 16 bajtový blok byl odesílán zařízením odečítajícím data z elektroměru (v tomto případě PC program SmartSet) jako součást sekvence zpráv, které zajišťovaly úvodní sestavení komunikačního kanálu s měřícím zařízením (viz kapitola 2.2: Asociace s elektroměrem). Tabulka 20 obsahuje hodnoty zmíněného 16 bajtového bloku získané že čtyř zaznamenaných komunikací.

Jak již bylo zmíněno, výrobce popis aplikačních dat nezveřejnil a ani hlubší analýza těchto čtyř bloků nedokázala odhalit jakýkoliv vztah mezi jejich obsahem nebo pravidlo, podle kterého byly tyto bloky sestaveny. Z dosavadních poznatků v této fázi práce tak nebylo jisté, zda bude spojení s elektroměrem možné navázat. 12 bajtové bloky proměnlivých hodnot byly odesílány elektroměrem jako součást poslední zprávy asociace obou zařízení a pravidlo pro jejich sestavení také nebylo známo. Tyto bloky však nepodstoupily žádnou další analýzu, neboť nebyl znám žádný způsob jejich využití

při odečítání hodnot z elektroměru.

Zpráva	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1.	C1	4A	CF	88	E5	5F	1D	FA	EF	92	01	19	96	D6	4D	D2
2.	45	4B	48	D5	8D	27	5E	7B	F1	02	75	BF	A7	7D	29	1A
3.	8C	35	7B	1B	A2	97	7A	92	30	D0	F4	FE	3F	2A	DC	21
4.	91	B8	29	3B	44	9E	B2	83	27	A4	D9	04	C5	6F	AC	BE

Tabulka 20: Proměnlivé datové bloky při navazování spojení

Posledním krokem při analýze zaznamenaných dat bylo zjištění významu jednotlivých bajtů s ohledem na standard DLMS/COSEM. Nejdříve byly vymezeny hlavičky jednotlivých protokolů (HDLC, MSDU) a poté i jednotlivá pole, které obsahují (adresy, kontrolní součty atd.). Výsledky analýzy jsou popsány v následující kapitole a ukázka zaznamenané komunikace jsou obsaženy v příloze této práce.

3.4.3 Výsledky analýzy aplikačních dat

Zaznamenání a analýza aplikačních dat byla důležitá součást procesu vývoje aplikace, neboť při ní byly zjištěny skutečnosti, které nebylo možné zjistit jinou metodou:

1. Fáze navázání komunikace s elektroměrem (asociace)
    - Elektroměr Kaifa pracuje ve zrychleném režimu a neodesílá finální potvrzení navázání spojení. Namísto toho již od klienta očekává HDLC komunikaci.
    - Klient se s elektroměrem asociuje postupným odesláním pěti zpráv, jejichž obsah je z většiny statický, a tedy snadno napodobitelný, avšak způsob vytvoření proměnlivé části jedné ze zpráv není známý.
2. Keepalive zprávy
    - Obě zařízení si po jistých časových intervalech vyměňují krátké statické zprávy, které svým chováním odpovídají keepalive zprávám známým z jiných protokolů. Nutnost odesílání těchto zpráv vyvíjenou aplikací bude muset být ověřena.
3. Odečtení údaje o spotřebě
    - Operace odečtení celkové naměřené spotřeby se skládá celkem ze čtyř zpráv, po dvou od každého zařízení
    - V první zprávě od klienta je viditelný OBIS kód. V odpovědi na tuto zprávu elektroměrem je obsažena hodnota naměřené spotřeby ve čtyř-bajtové reprezentaci, jejíž první bajt se nachází na 18. pozici.
    - Význam zbylých dvou zpráv není znám, jejich obsah je však statický.
4. Ukončení spojení
    - Pro odpojení od elektroměru je nutné odeslat krátkou (10 B) zprávu se statickým obsahem, která je následně potvrzena poslední zprávou ze strany elektroměru. Tímto je spojení ukončeno.
5. Ostatní poznatky:
    - HDLC komunikace probíhá rychlostí 9600 Bd
    - Elektroměr Kaifa používá v HDLC rámcích rozšířené adresní pole

serveru na 2 bajty (viz kapitola 2.3.3: Adresní pole). Hlavička má poté celkem 8 bajtů.

Tyto poznatky byly použity při programování části aplikace, jež obstarává komunikaci s elektroměrem přes sériovou linku a její vývoj je popsán v části 3.5.4.

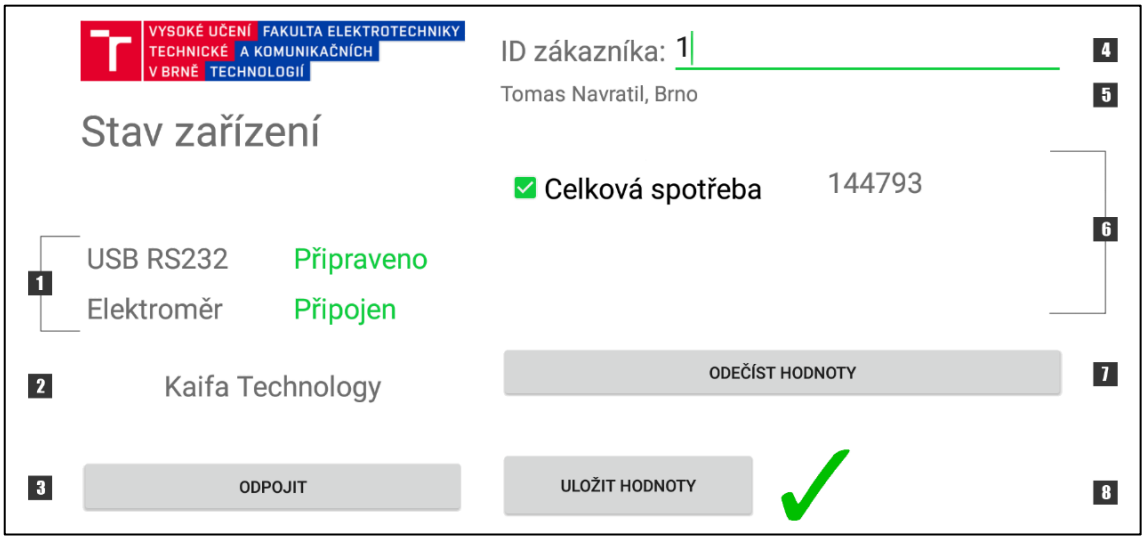
3.5 Vývoj aplikace

Aplikace pro OS Android byla vyvíjena pomocí nástroje Android Studio (verze 2.2) běžícím nad operačním systémem MS Windows 7 a 10. Před samotným vývojem bylo nutné rozhodnout, kterou verzi API bude aplikace podporovat. Zvoleno bylo API 22, které odpovídá OS Android 5.1, tedy přesně verzi, která byla nainstalována na testovací tablet. S tímto nastavením je vyvinutá aplikace kompatibilní s cca 23.2% [12] zařízeními, na kterých je OS Android nainstalován. Aplikace byla naprogramována v jazyce Java bez podpory C++.

3.5.1 Uživatelské rozhraní

Vzhledem k použití tabletu bylo uživatelské rozhraní navrženo tak, aby se používalo zobrazení na šířku (landscape mode). Toto nastavení nemůže uživatel obejít a aplikace se do něj automaticky přepne při spuštění nezávisle na aktuální orientaci zařízení. Kromě lepšího využití plochy displeje toto nastavení zpříjemňuje práci s USB kabelem optického převodníku, který je v tomto případě vyveden na hraně tabletu vpravo od displeje namísto pod displejem v případě použití zobrazení na výšku.

Samotné uživatelské rozhraní bylo rozděleno do dvou částí. Levá polovina slouží k zobrazení stavových údajů o připojených zařízeních a k ovládání spojení mezi tabletem a elektroměrem. Pravá polovina je vyhrazena pro operace odečítání dat z elektroměru a jejich ukládání do databáze.



Obrázek 12: Uživatelské rozhraní vyvíjené aplikace

V následujícím seznamu je popsán význam jednotlivých položek očíslovaných ve screenshotu z aplikace (viz Obrázek 12):

1. Na tomto místě je zobrazen stav připojených zařízení. Konkrétně informace, zda je detekována sériová linka připojená do USB OTG portu a stav logického spojení s elektroměrem.
2. Při navazování logického spojení s elektroměrem je na toto místo vypsán název výrobce elektroměru, který je získán z první datové zprávy odeslané elektroměrem (viz kapitola 3.5.11).
3. Tímto tlačítkem je ovládáno spojení s elektroměrem. Ve výchozím stavu je zobrazen popisek „PŘIPOJIT“ a jeho stisknutím se odstartuje připojení k elektroměru. Po úspěšném připojení je popisek změněn na „ODPOJIT“ a po jeho opětovném stisknutí je tablet odpojen od elektroměru.
4. Do tohoto pole je uživatelem vkládáno identifikační číslo zákazníka, u kterého je prováděn odečet údajů. Zadávání hodnot je omezeno pouze na čísla a po potvrzení zadané hodnoty na softwarové klávesnici je provedeno ověření existence zákazníka v databázi a v případě pozitivního výsledku jsou z databáze staženy zákazníkovy základní údaje.
5. Stažené údaje o zákazníkovi jsou zobrazeny na tomto místě.
6. Na této pozici je zobrazen zaškrtávací seznam s názvy hodnot, které je možné z elektroměru odečíst. K výběru je pouze možnost „Celková spotřeba“, návrh však počítá se snadným rozšířením tohoto seznamu o další položky. Po odečtení požadovaných položek je vpravo od názvu položky zobrazena odečtená hodnota.
7. Stisknutím tohoto tlačítka se spustí odečítání hodnot položek zaškrtnutých v seznamu (6.).
8. Po odečtení hodnot je aktivováno tlačítko pro uložení těchto hodnot do databáze. Vpravo od tlačítka je během ukládání zobrazen progress bar, který je po dokončení operace nahrazen ikonou indikující úspěšné dokončení ukládání dat.

3.5.2 Ovládání sériové linky z OS Android

Přestože je hardware operačním systémem podporován, neobsahuje žádné nástroje pro ovládání sériové linky. Protože se nejedná o primární téma této práce, bylo přistoupeno k použití knihovny třetí strany. Na základě pozitivních referencí a široké podpory hardwaru byla vybrána knihovna **UsbSerial** [13]. Ta je vyvíjena španělským vývojářem specializujícím se na OS Android Felipem Herranzem, který ji uvolnil pod MIT licencí. Knihovna je tedy volně šiřitelná bez poplatků i v případě komerčního použití.

3.5.3 Použití UsbSerial knihovny

Veškeré operace se sériovou linkou se provádí pomocí knihovní třídy `UsbService`. Po instanciaci objektu této třídy je ihned spuštěna stejnojmenná služba, která dále běží na pozadí. Úkolem této služby je zajistit oprávnění pro přístup k USB portu, monitorovat k němu připojená zařízení a provádět kontrolu přítomnosti hardwaru, který podporuje standard RS-485. Služba reaguje na jakékoliv nově připojené zařízení a po kontrole jeho vlastností informuje hlavní aplikaci pomocí jedné z pěti definovaných servisních zpráv zobrazených v Tabulce 21.

Název zprávy	Význam
ACTION_USB_PERMISSION_GRANTED	Uživatel udělil oprávnění pro přístup k USB portu. Zařízení je připraveno k použití.
ACTION_NO_USB	Žádné zařízení není připojeno.
ACTION_USB_DISCONNECTED	Zařízení bylo připojeno, ale uživatel jej odpojil nebo došlo k chybě na fyzické vrstvě.
ACTION_USB_NOT_SUPPORTED	Připojené zařízení neobsahuje RS-485 komponentu nebo není podporována.
ACTION_USB_PERMISSION_NOT_GRANTED	Uživatel oprávnění neudělil. Zařízení není možné použít.

Tabulka 21: Zprávy služby UsbService

Protože se jedná o službu, její běh je z pohledu OS oddělen od hlavní aplikace. Pro zasílání servisních zpráv hlavní aplikaci tedy bylo nutné využít zabudovaný mechanismus OS Android pro komunikaci mezi procesy. Služba pomocí metody `sendBroadcast()` odesílá přes operační systém objekty třídy `Intent` [23], které obsahují identifikaci zprávy v podobě textového řetězce (`String`). V hlavní aplikaci je vytvořen objekt třídy `BroadcastReceiver`, jehož povinná metoda `onReceive()` je zavolána operačním systémem vždy, když je přes něj přenášen objekt `Intent`. Aby aplikace nemusela zpracovávat zprávy, které pro ni nejsou určeny, je `BroadcastReceiver` doplněn filtrem, který specifikuje typy objektů `Intent`, které služba vysílá. Dle tohoto typu je poté uživatel přes GUI informován o stavu připojeného zařízení. Následující ukázka zdrojového kódu obsahuje zachycení zprávy, která potvrzuje získání oprávnění k přístupu k USB portu aplikací:

```
private final BroadcastReceiver mUsbReceiver = new BroadcastReceiver() {
  @Override
  public void onReceive(Context context, Intent intent) {
    switch (intent.getAction()) {
      case UsbService.ACTION_USB_PERMISSION_GRANTED:
      …
      break;
      …
} } };
```

Parametry sériové linky se ovládají pomocí metod třídy UsbSerialDevice. Její instance je součástí objektu UsbService. Parametry lze měnit libovolně i v průběhu již funkčního připojení, což je prerekvizita pro úspěšné navázání spojení dle DLMS/COSEM protokolu (2.2.1). Pro nastavení parametrů jsou použity metody, jejichž popis je uveden v Tabulce 22.

Metoda	Význam
`setBaudRate(int baudRate)`	Rychlost linky [Bd]
`setDataBits(int dataBits)`	Počet datových bitů
`setStopBits(int stopBits)`	Počet stop bitů
`setParity(int parity)`	Parita
`setFlowControl(int flowControl)`	Řízení toku dat

Tabulka 22: Metody pro nastavení parametrů sériové linky

K odeslání dat přes sériovou linku je určena metoda `write(…)`. Data k odeslání jsou předávána parametrem datového typu pole bajtů. Další zpracování je v režii knihovny.

Příchozí data jsou ukládána do vyrovnávací paměti o velikosti 16 kB. Tato paměť je v pravidelných intervalech vyprazdňována a její obsah je předáván hlavní aplikaci v podobě zpětného volání (call back). Pro zpracování zpětného volání je v hlavní aplikaci připraven ovladač (handler). Při zpětném volání jsou jeho metodě `handleMessage(…)` předána přijatá data opět v podobě pole bajtů. V této metodě jsou data následně zpracována (viz kapitola 3.5.8: Uložení přijatých dat).

3.5.4 Komunikace s elektroměrem

Pro práci s elektroměrem připojeným přes sériovou linku byla navržena třída `ElektromerConnection`, která je součástí třídy `UsbSerivce`. Z důvodu časové náročnosti operací prováděných nad sériovou linkou je třída `ElektromerConnection` zavedena jako rozšíření abstraktní třídy `AsyncTask` [21]. Díky tomu je běh veškerých operací se sériovou linkou prováděn na pozadí v novém vlákně a uživatelské rozhraní aplikace zůstává responsivní. Pokud by operace se sériovou linkou nebyly prováděny asynchronně vzhledem k hlavnímu vláknu aplikace, s uživatelským rozhraním by nebylo možné pracovat do ukončení operace, což v případě několikasekundových požadavků není akceptovatelné.

Třída `AsyncTask` obsahuje několik metod pro asynchronní programování. Základní z nich je definována následujícím prototypem:

```
protected Object doInBackground(Object[] params)
```

Jak bylo zmíněno výše, kód obsažený v implementaci této metody je prováděn „na pozadí“, tedy ve svém vlastním vlákně. Při implementaci je nutné metodu „přepsat“ za pomoci značky `@Override`. V implementaci třídy `ElektromerConnection` byl uvnitř zmíněné metody naprogramován rozhodující mechanismus, který na základě hodnoty proměnné `action` provede se sériovou linkou požadovanou operaci. Hodnota proměnné `action` je metodě předána pomocí parametru typu `Integer`. V současnosti jsou podporovány tři operace popsané v Tabulce 23.

Kód operace	Význam
ACTION_SENS_CON	Připojení k elektroměru
ACTION_SENS_DISCO	Odpojení od elektroměru
ACTION_GET_APLUS	Získání údaje o naměřené spotřebě

Tabulka 23: Operace se sériovou linkou

Na základě zvolené operace jsou následně přes sériovou linku odeslána data, která byla získána při analýze aplikačních dat. Tato data jsou uložena ve statických polích uvnitř třídy `ElektromerConnection`. Před provedením kterékoliv operace je nejdříve nastavena hodnota proměnné `BUSY` na `TRUE`. Tato proměnná je deklarována ve třídě `UsbSerivce` a je díky ní možné z hlavní aplikace kontrolovat stav sériové linky. Tímto mechanismem je zabráněno odesláním několika souběžných požadavků na sériovou linku. Po dokončení operace je hodnota proměnné nastavena na `FALSE`.

Pro započetí asynchronní operace je vždy nutné vytvořit novou instanci třídy `ElektromerConnection`. K tomu slouží metoda Obsluha ve třídě UsbService deklarovaná následujícím prototypem:

```
public void Obsluha(int action, int param)
```

První parametr slouží k rozlišení zdroje, který má operaci zpracovat. Nyní je podporována pouze sériová linka označená kódem `ACTION_SERIAL`. V tomto případě je vytvořena nová instance třídy `ElektromerConnection`, které je předána hodnota parametru `param`. Ten obsahuje označení jedné z výše zmíněných operací, která se se sériovou linkou provede. Následující část kódu obsahuje spuštění asynchronního vlákna:

```
public void Obsluha(int action, int param) {
  switch (action) {
    case ACTION_SERIAL:
      new ElektromerConnection(param).execute();
      break;
} }
```

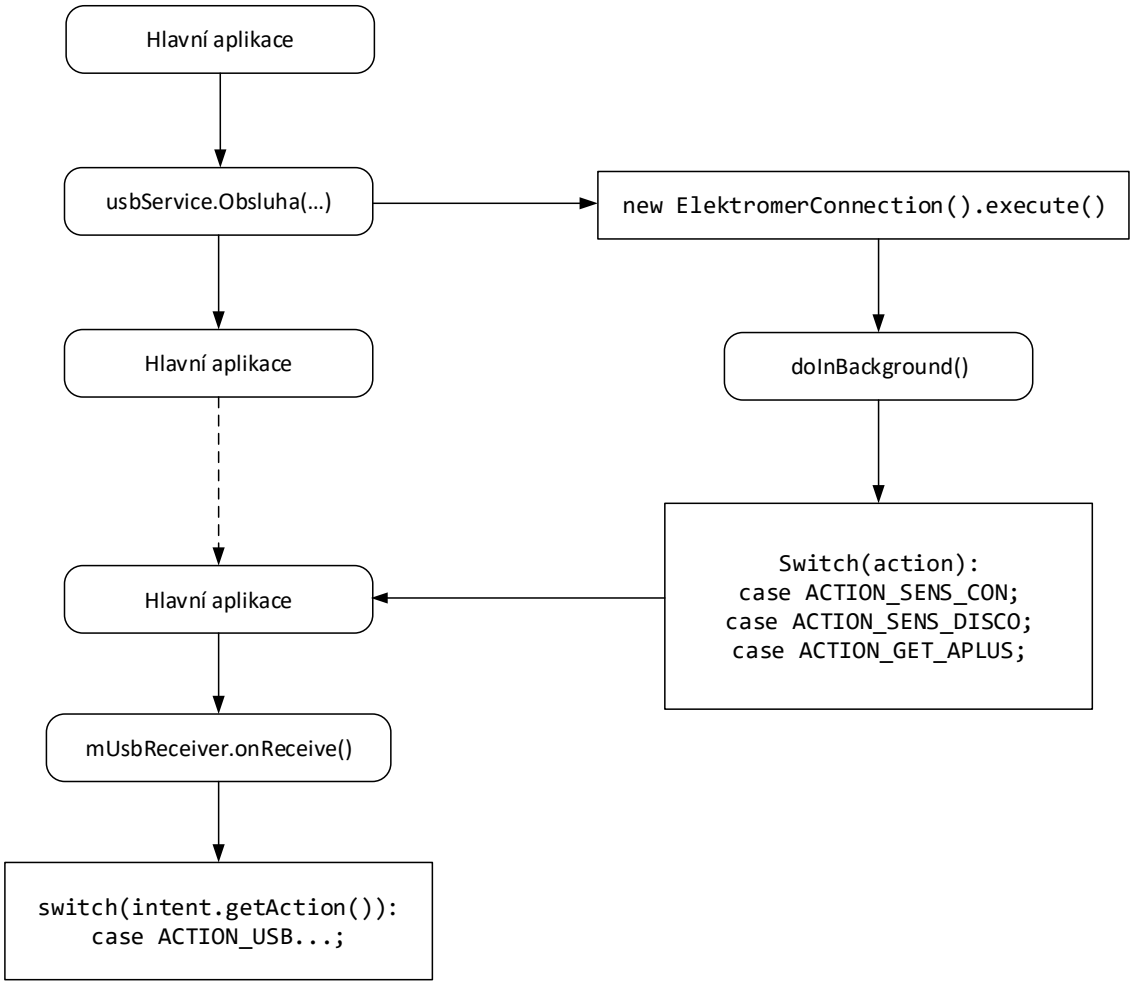
Odesílání dat přes sériovou linku zajišťuje metoda `sendData`:

```
private void sendData(byte[] data, int waitTime) {
  if (serialPort == null) return;
  serialPort.write(data);
  try {
    Thread.sleep(waitTime);
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
  serialPort.read(mCallback);
}
```

Metoda obsahuje kontrolu stavu sériového portu a je volána se dvěma parametry:

- `data:`
  Jedná se o pole bajtů, které mají být poslány přes sériovou linku.
- `waitTime:`
  Tento parametr udává dobu v milisekundách, kterou je potřeba vyčkat po odeslání dat na sériovou linku na příjem odpovědi od elektroměru. Na tuto dobu je celé vlákno uspáno metodou sleep(). V případě neuspání vlákna by aplikace odeslala všechny zprávy ihned po sobě, což by v případě použitého modelu komunikace dotaz-odpověď vedlo k chybám v komunikaci.

Pro lepší přehled je celý proces ovládání sériové linky zobrazen ve zjednodušené formě na Obrázku 13.



Obrázek 13: Schéma procesu obsluhy sériové linky

3.5.5 Operace připojení k elektroměru

Započetí této operace je vázáno na stisk tlačítka „PŘIPOJIT“ v uživatelském rozhraní. Povel je přes metodu `Obsluha` předán objektu třídy `ElektromerConnection`, kde je

spuštěna akce „ACTION_SENS_CON“. Zprávy, které se budou v průběhu navazování spojení s elektroměrem přes sériovou linku odesílat, jsou součástí souboru staticky definovaných polí bajtů uvnitř třídy `ElektromerConnection` a nesou označení „`hello[1-5]`“. Tyto zprávy jsou během navazování spojení postupně odesílány vždy se staticky určenou čekací dobou na odpověď od elektroměru. Tuto dobu je možné vypočítat na základě nastavení sériové linky a počtu odesílaných bitů. V případě sériové linky rychlost v baudech odpovídá rychlosti v kb/s. Nutné je však zjistit počet servisních bitů (stop a paritní) přidaných k datovým. Při nastavení linky 8N1 jsou k 8 datovým bitům přidány další 2 (start bit a stop bit) a pro zjednodušení je možné říci, že 1 bajt je odeslán na 10 bitech. S parametry 7E1 jsou k 7 datovým bitům přidány další 3 (start bit, paritní bit, stop bit). Doba odesílání prvotních zpráv je vypočítána v Tabulce 24. Doba čekání však byla úspěšně nastavena až při testování aplikace. Při něm bylo zjištěno, že elektroměru Kaifa je nutné požadavky zasílat ihned po přijetí jeho odpovědí, avšak zároveň je nutné zajistit, aby odesílání nového požadavku nezačalo před dokončením příjmu předchozí odpovědi, což při testování vždy vedlo k ukončení komunikace. Délka čekání je tak pro každou zprávu jiná.

Název pole	Velikost [B]	Rychlost linky [Bd]	Parametry linky	Doba [ms]	Nastavená doba [ms]
hello1	5	300	7E1	160	1300
hello2	6	300	7E1	230	500
hello3	35	9600	8N1	36	500
hello4	91	9600	8N1	95	700
hello5	47	9600	8N1	49	500

Tabulka 24: Vypočítaná doba odesílaných zpráv

Před odesláním prvních dvou zpráv jsou nastaveny parametry sériové linky dle DLMS/COSEM standardu (viz kapitola 2.2: Asociace s elektroměrem) na 7E1 a rychlost 300 Bd. Po přijetí odpovědi na druhou ze zpráv jsou parametry linky upraveny na 8N1 a rychlost 9600 Bd. Tuto rychlost dle standardu určuje měřící zařízení a je odeslána v jeho první zprávě jako parametr „Z“. Elektroměr Kaifa na této pozici odesílá hodnotu „5“, což odpovídá právě rychlosti 9600 Bd. Tou jsou odeslány zbylé tři zprávy navazování komunikace a veškerá pozdější komunikace. Protože jsou tyto parametry nastaveny přímo na sériovém portu, není nutné je znovu nastavovat po vytvoření nové instance třídy `ElektromerConnection`.

3.5.6 Operace získání hodnoty naměřené spotřeby

Odečíst jakékoliv hodnoty z elektroměru je možné až po úspěšném navázání komunikace. Následně je v uživatelském rozhraní nutné zadat identifikační číslo (ID) zákazníka, kterému elektroměr patří, a vyčkat na stažení údajů o zákazníkovi z databáze. Pokud v této fázi není k dispozici připojení k internetu, není možné hodnoty odečíst. V případě úspěšného stažení údajů o zákazníkovi a jejich zobrazení v GUI je možné vybrat položku „Celková naměřená spotřeba“ a stisknout tlačítko „Odečíst hodnoty“.

Tímto je odstartována operace stažení hodnoty spotřeby z elektroměru.

Tato operace pod označením „ACTION_GET_APLUS“ je součástí třídy `ElektromerConnection` a sestává z odeslání dvou předem definovaných zpráv o celkové dálce 56 bajtů. Obsah těchto dvou zpráv byl získán analýzou zaznamenaných dat ze softwaru SmartSet (viz kapitola 3.4). Hodnota naměřené spotřeba je získána z dat přijatých od elektroměru v hlavní aplikaci (viz kapitola 3.5.11).

Po odečtení hodnot již není dovoleno operaci opakovat a pro opakované odečtení je nutné se od elektroměru nejdříve odpojit a následně znovu připojit.

3.5.7 Odpojení od elektroměru

V kterékoliv fázi práce s elektroměrem je možné spojení ukončit. Tuto operaci iniciuje uživatel stisknutím tlačítka „Odpojit“ v GUI. Toto tlačítko je dostupné pouze v případě, že bylo spojení s elektroměrem úspěšně navázáno. Po stisknutí tlačítka je odstartována operace třídy `ElektromerConnection` označená kódem „ACTION_SENS_DISCO“. Aplikace odešle jednu 10 bajtovou zprávu s požadavkem na ukončení spojení, kterou elektroměr potvrdí poslední zprávou v rámci dosavadní komunikace o délce 35 bajtů. Tímto je spojení ukončeno.

Součástí této operace jsou i události v hlavní aplikaci. Z uživatelského rozhraní jsou odstraněny všechny údaje týkající se zákazníka a naměřených hodnot. V aplikaci je smazán obsah vektoru přijatých dat `receivedBytes` a všechny pomocné proměnné jsou nastaveny do výchozího stavu odpovídajícího stavu při spuštění aplikace.

3.5.8 Uložení přijatých dat

Při prvotním testování komunikace bylo zjištěno, že obsah vyrovnávací paměti pro přijatá data je aplikaci odesílán ve velmi krátkých intervalech. Při nastavení rychlosti linky na 300 Bd bylo následkem odesílání jednotlivých přijatých bajtů. Pro zpracování přijatých dat bylo nutné uchovat veškerá přijatá data v hlavní aplikaci a jejich zpracování podřídit jiným událostem, než je samotné přijetí dat. K tomuto účelu byl vytvořen vektor bajtů `receivedBytes`. Veškerá přijatá data jsou do tohoto vektoru ukládána po jednotlivých bajtech:

```
@Override
public void handleMessage(Message msg) {
  switch (msg.what) {
    case UsbService.MESSAGE_FROM_SERIAL_PORT:
      if (receivedBytes == null) return;
      byte[] data = (byte[]) msg.obj;
      for (byte b : data) {
         receivedBytes.add(b);
      }
      manageReceivedData();
      break;
} }
```

3.5.9 Zpracování přijatých dat

Pro zpracování dat byla v hlavní aplikaci naprogramována metoda `manageReceivedData()`. Ta je volána vždy, když jsou zpětným voláním přijata data ze sériové linky. Metoda pracuje výhradně nad vektorem přijatých bajtů `receivedBytes`. Typ operace provedené nad přijatými daty je určen hodnotou proměnné `GET` typu `int`.

Název	Hodnota	Význam
--	0	Neprovede se žádná operace
GET_VENDOR	1	Získá název výrobce elektroměru
GET_APLUS	2	Získá hodnotu naměřené spotřeby el. energie

Tabulka 25: Možné hodnoty proměnné GET

Hodnota proměnné `GET` je kontrolována při každém zavolání metody `manageReceivedData()` a v případě, že je odlišná od hodnoty 0, je nad přijatými daty provedena požadovaná operace (viz Tabulka 25). Po dokončení požadované operace je proměnná `GET` nastavena zpět na hodnotu `0`.

3.5.10 Operace GET_VENDOR

Tato operace je iniciována při navazování každého nového spojení, během kterého je údaj o výrobci elektroměrem odesílán v odpovědi na žádost o připojení.

Pro získání názvu připojeného elektroměru byla vytvořena metoda `getDeviceName()`. Ta je volána metodou pro zpracování přijatých dat, pokud vektor přijatých bajtů splňuje následující tři podmínky:

1. Obsahuje více než 30 bajtů
2. Obsahuje znak CR (\r)
3. Obsahuje znak LF (\n)

Nastavení těchto podmínek vyplývá z obsahu přijatých zpráv při sestavování spojení s elektroměrem přes sériovou linku (2.2.1). První znak identifikačního řetězce se nachází na 7. pozici první přijaté zprávy a je ukončen symboly CR a LF. Textový řetězec mezi těmito pozicemi získá právě metoda `getDeviceName()`. Ta sestaví název ze znaků nacházejících se na 7. pozice a dále až po znak předcházející symbol CR. Textový řetězec obsahující získaný název je poté předán k zobrazení v uživatelském rozhraní.

3.5.11 Operace GET_APLUS

Jedná se operaci, která z přijatých dat získá číselnou hodnotu odpovídající celkové naměřené spotřebě elektrické energie. Operace je iniciována vybráním patřičné volby v uživatelském rozhraní a stisknutím tlačítka „Odečíst údaje“. Hodnota je poté extrahována z dat přijatých z elektroměru. Protože však data (payload) přijímaná

z elektroměru nemají obsah určen standardem a jejich význam je dán výrobcem elektroměru, který je veřejně neposkytuje, bylo nejdříve nutné zjistit význam přijímaných dat, konkrétně pozici, na které se vyskytuje první bajt hodnoty celkové spotřeby.

Operace ke své činnosti kromě vektoru přijatých dat vyžaduje další dvě proměnné:

- dataOffset:

  Před odečtením údaje o spotřebě je nejdříve nutné navázat spojení s elektroměrem. Během navazování spojení jsou do vektoru přijatých dat ukládána data a je tedy nutné mít uloženou pozici, která předchází datům přijatým v rámci odpovědi na požadavek o odečtení údaje o spotřebě. Hodnota proměnné `dataOffset` je určena z délky vektoru přijatých dat jako první krok při odesílání požadavku na odečtení údaje.

- dataBuffer:

  Pomocný vektor bajtů `dataBuffer` je použit pro dočasné uložení 4 bajtů, které obsahují hodnotu naměřené spotřeby reprezentovanou datovým typem „`unsigned long integer`“, tedy 32 bitové bezznaménkové číslo.

Převod hodnoty započne, jakmile velikost vektoru receivedBytes přesáhne hodnotu [dataOffset + 30]. Touto podmínkou je zaručeno, že již bajty obsahující požadovanou hodnotu byly přijaty a uloženy do vektoru přijatých bajtů. Prvním krokem extrakce hodnoty je zkopírování zmíněných 4 bajtů reprezentující naměřenou hodnotu do pomocného vektoru dataBuffer. Analýzou aplikačních dat bylo zjištěno, že pozice prvního bajtu se nachází na indexu [dataOffset + 17]. Po dokončení kopírování do pomocného vektoru je tento vektor převeden na prosté pole bajtů, které je pomocí metody unsignedIntToLong() převedeno na výslednou celočíselnou hodnotu. Získaná hodnota je zobrazena v GUI a zároveň uložena do proměnné readData.aplus.

ReadData je instance třídy DataToSave:

```
public class DataToSave {
  public int cid      = 0;
  public String  aid = "";
  public long aplus   = 0;
}
```

Tato třída byla navržena pro předávání hodnot mezi hlavní aplikací a třídou obsluhující HTTP požadavky pro přístup k SQL API. Obsahuje atributy pro uložení identifikačního čísla zákazníka (cid), Android ID zařízení, které odečet hodnoty z elektroměru provedlo, a atribut pro uložení naměřené hodnoty. V případě potřeby ukládat další údaje je tak mechanismus předávání dat snadno rozšířitelný o další hodnoty.

3.5.12 Práce s databází

Aplikace komunikuje s databází přes API, které přijímá příkazy v podobě HTTP dotazů GET (viz kapitola 3.3). K tomuto účelu byla naprogramována třída `HttpReq`, která odesílá požadavky i zpracovává odpovědi. Tyto operace s databází jsou časově náročné

a během jejich zpracovávání by nebylo GUI aplikace responzivní. Proto byla třída `HttpReq` naprogramována jako rozšíření abstraktní třídy `AsyncTask` a její instance jsou tak spouštěny ve vlastním vlákně. Objektům této třídy jsou při jejich vytvoření předány konstruktorem dvě hodnoty:

1. `context`: jedná se o kontext hlavní aplikace pro zpětnou komunikaci
2. `action`: číselná proměnná označující operaci, která se má s DB provést

Proměnná `action` může nabývat dvou hodnot, které jsou staticky definovány ve třídě `HttpReq`, a jejich význam je popsán v následující Tabulka 26:

Označení	**Hodnota**	**Význam**
HTTP_REQ_INFO	1	Získání údajů o zákazníkovi
HTTP_SAVE_DATA	2	Uložení hodnot do databáze

Tabulka 26: Operace třídy HttpReq

Hodnoty obou proměnných jsou uloženy ve stejnojmenných atributech objektu kvůli jejich využití při sestavování HTTP požadavku a po přijetí odpovědi od API i k sestavení odpovědi, která je následně odeslána zpět hlavní aplikaci.

V případě potřeby komunikovat s databází jsou objekty této třídy vytvářeny v hlavní aplikaci bez referenční proměnné tímto způsobem:

```
new HttpReq(getApplicationContext(), HttpReq.HTTP_REQ_INFO).execute(cid);
```

Z tohoto příkladu je i patrný způsob předávání dat, se kterými nálsedně pracuje databáze. Ta jsou předána pomocí metody `execute()`, které je možné zadat libovolný počet objektů jako její parametry. V tomto případě se jedná o odeslání ID zákazníka, o němž mají být z databáze staženy informace. V případě zápisu do databáze je na jeho místě předán objekt třídy `DataToSave`.

V tomto případě vytvoření objektu je ihned volána metoda `doInBackground()`, v níž se nachází obslužný kód pro HTTP dotazy. Obě operace mají společnou první část, ve které je definována URL adresa SQL API a pomocné proměnné, které slouží k parsování XML dokumentu, který je API vrácen jako odpověď:

```
String baseUrl = "http://navtom.cz/dp/api.php?";
URL url;
DocumentBuilderFactory dbf;
DocumentBuilder db;
Document doc;
```

Následně je program větven podle hodnoty proměnné `action`. V případě požadavku na získání údajů o zákazníkovi je sestavena URL obsahující zákazníkovo ID a pomocí metody `openStream()` je získán obsah dokumentu, na který URL adresa odkazuje. Tento obsah je vrácen jako tzv. stream, tedy sekvence bajtů. Z této sekvence je sestaven nový DOM dokument pomocí metody parse() třídy DocumentBuilder. Výsledný XML dokument je poté získán normalizací DOM dokumentu. Ta se provádí pro korektní převod DOM elementů na XML prvky, tedy např. správný převod víceřádkových hodnot jednotlivých XML elementů [22]. Ze získaného XML dokumentu jsou poté vybrány všechny prvky (elementy), které jsou pojmenovány jako „info“. Protože bylo API naprogramováno tak, že vrací pouze jeden prvek tohoto názvu, je přímo přistoupeno

k prvnímu získanému „info“ prvku na indexu 0 v seznamu `infoList`. Obsah prvku je převeden na text a následně je rozhodnuto, zda obsahuje požadovanou informaci. Pokud je řetězec prázdný nebo došlo k výjimce při parsování XML dokumentu (tj. neobsahuje jediný prvek nazvaný „info“), informace o zákazníkovi se nepodařilo získat, proměnná `found` je nastavena na hodnotu `FALSE` a do řetězce `info` je uložena chybová hláška. V ostatních případech je získaný text uložen taktéž do řetězce `info`, ale proměnná `found` je nastavena na hodnotu `TRUE`. V obou případech je poslední krokem uložení řetězce `found` do univerzálního objektu `toSend`, který je spolu s proměnnou `found` dále zpracován v části třídy zajišťující komunikaci s hlavní aplikací.

Druhým použitým požadavkem na operaci s databází je uložení naměřených hodnot. Příprava XML dokumentu je shodná s operací pro získání údajů o zákazníkovi kromě části sestavující URL adresu. Ta obsahuje hodnotu q=w a další tři údaje získané z objektu toSend – tj. ID zákazníka, identifikační kód odečítajícího zařízení a hodnotu naměřené spotřeby elektrické energie.

```
url = new URL(baseUrl + String.format("q=w&cid=%d&aid=%s&val=%d",
data.cid, data.aid, data.aplus));
```

Po zkompletování XML dokumentu jsou vybrány všechny prvky s názvem „body“. Opět je předpokládáno, že API vrátí pouze jeden prvek tohoto názvu v celém XML dokumentu, a proto je přímo přistoupeno k prvnímu z nich na indexu 0. Pokud tento proces vyvolá výjimku nebo prvek „body“ obsahuje text „ERR“ je proces ukládání dat považován za neúspěšný a hodnoty proměnné dataSaved je nastavena na FALSE. Jinak je ponechána její výchozí hodnota TRUE.

Po vykonání metody doInBackground() jsou připravena data pro odeslání do hlavní aplikace. To zajišťuje metoda onPostExecute(), která je automaticky volána právě po provedení všech příkazů v metodě doInBackground() [21]. Protože metody třídy HttpReq jsou prováděny asynchronně vůči hlavní aplikaci, není z hlavního vlákna možné kontrolovat vnitřní chod instancí této třídy. K předání dat hlavní aplikaci bylo využito techniky odesílání krátkých mezivláknových zpráv, tzv. Intents [23]. Odeslání takovéto zprávy sestává ze tří kroků:

1. Vytvoření nové instance třídy Intent
2. Předání dat vytvořené instanci
3. Odeslání zprávy

Při vytváření nové instance je nutné zadat unikátní jméno nové zprávy. Toto jméno je použito k rozlišení různých typů zpráv a hlavní aplikace podle něj filtruje zprávy, které jsou pro ni určeny. Názvy jsou standardní textové řetězce a ve třídě `HttpReq` jsou staticky definovány tři, viz Tabulka 27. Jejich názvy kvůli lepší orientaci v kódu odpovídají akcím, které jsou požadovány z hlavní aplikace (Tabulka 26).

Název řetězce	Hodnota
HTTP_INFO_READY	cz.navtom.elektromer.HTTP_INFO_READY
HTTP_SAVE_OK	cz.navtom.elektromer.HTTP_SAVE_OK
HTTP_SAVE_ERR	cz.navtom.elektromer.HTTP_SAVE_ERR

Tabulka 27: Názvy pro Intents

O typu připravované zprávy je rozhodnuto dle hodnoty proměnné `action`. V případě stahování údajů o zákazníkovi je typ zprávy „HTTP_INFO_READY“, ke které jsou navíc přidány hodnoty proměnných `toSend` a `found`. V případě ukládání hodnot je na základě hodnoty pomocné proměnné dataSaved indikující úspěšnost ukládání dat vytvořena zpráva typu „HTTP_SAVE_OK“ nebo „HTTP_SAVE_ERR“. Po vytvoření je zpráva odeslána metodou sendBroadcast(), která musí být volána v kontextu hlavní aplikace. Ten je získán při vytvoření instance třídy `HttpReq` z hlavní aplikace.

```
intent = new Intent(HTTP_INFO_READY);
context.sendBroadcast(intent);
```

Odeslaná zpráva je přečtena v hlavní aplikaci metodou `onReceive()` objektu `mUsbReceiver`, který je instancí třídy BroadcastReceiver. Tento objekt je součástí knihovny UsbSerial použité pro komunikaci se sériovou linkou (viz kapitola 3.5.3) a byl dodatečně rozšířen o zpracování zpráv odesílaných objekty třídy HttpReq. K objektu `mUsbReceiver` je přiřazen filtr zpráv IntentFilter, jehož nastavení je provedeno v metodě `setFilters()` v hlavní aplikaci. K původním zprávám ze třídy UsbService byly přidány ty ze třídy HttpReq.

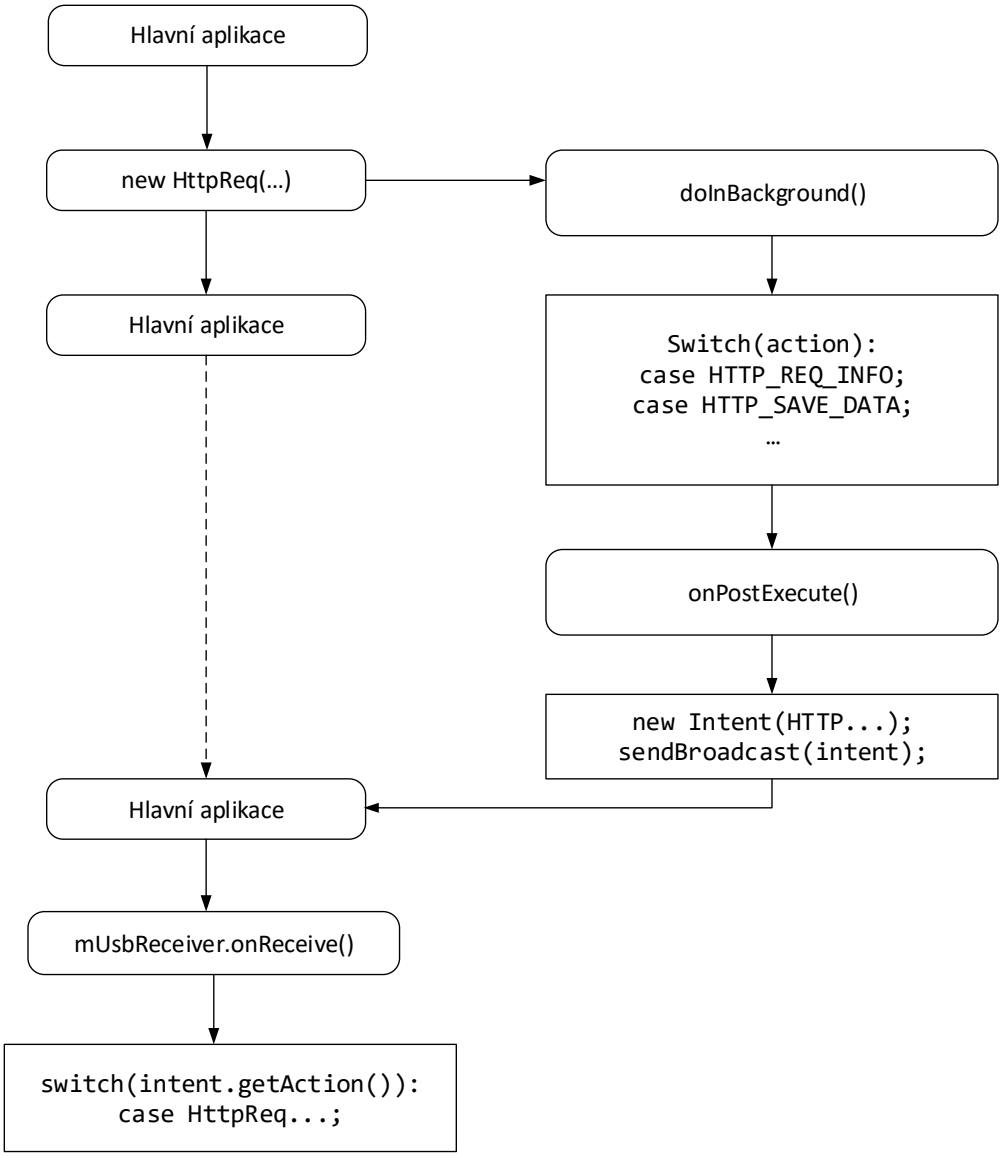
```
private final BroadcastReceiver mUsbReceiver = new BroadcastReceiver() {
  public void onReceive(Context context, Intent intent) {…}
}

private void setFilters() {
        IntentFilter filter = new IntentFilter();
        …
        filter.addAction(HttpReq.HTTP_INFO_READY);
        filter.addAction(HttpReq.HTTP_SAVE_OK);
        filter.addAction(HttpReq.HTTP_SAVE_ERR);

        registerReceiver(mUsbReceiver, filter);
}
```

V metodě `onReceive(…)` je provedeno větvění dle typu zprávy:

- HttpReq.HTTP_INFO_READY:
  Ze zprávy jsou extrahovány proměnné `info` a `found`. Dle hodnoty proměnné found je nastaveno ID zákazníka uvnitř objektu `readData` a v GUI je zobrazen textový řetězec `info`.
- HttpReq.HTTP_SAVE_OK:
  V GUI je zobrazena ikona potvrzující uložení informací do databáze.
- HttpReq.HTTP_SAVE_ERR:
  V GUI je zobrazena ikona značící neúspěšný pokus o uložení dat a je znovu povoleno tlačítko „Uložit hodnoty“.

V Obrázku 14 je zobrazeno schéma celého procesu komunikace s databází přes SQL API. Posledním nutným krokem k úspěšné komunikaci s databází bylo zajištění práv (permission) pro přístup k internetu aplikací. To bylo zajištěno přidáním následujícího řádku do souboru AndroidManifest.xml, který dovoluje využití jak Wi-Fi, tak i mobilních datových přenosů:

```
<uses-permission android:name="android.permission.INTERNET" />
```



Obrázek 14: Schéma komunikace s SQL API

3.6 Testování aplikace

Aplikace byla testována v průběhu vývoje vždy po dokončení jedné z etap. Cílem bylo ověření zamýšlené funkčnosti a ověření poznatků, které byly získány o protokolu DLMS/COSEM a analýzou aplikačních dat.

3.6.1 Navázání spojení

První etapou bylo naprogramování připojení k elektroměru (viz kapitola 3.5.5). Protože se jednalo o první případ komunikace s elektroměrem, byla nejdříve otestována funkčnost UsbSerial knihovny. To bylo provedeno bez připojení senzoru k elektroměru odesláním náhodných dat přes sériovou linku s výchozím nastavením. Po ověření aktivity

sériové linky, která je indikována zelenou LED diodou na optickém senzoru, byly parametry linky nastaveny na ty pro navázání spojení (viz Tabulka 1) a byla odeslána první zpráva pro sestavení spojení. Následně byla pozorována aktivita senzoru v příchozím směru. Příjem dat z elektroměru je indikován červenou LED diodou. Po ověření funkčnosti obousměrné aplikace bylo dále nutné otestovat schopnost knihovny měnit parametry linky během probíhající komunikace, což bylo potvrzeno po doprogramování metody pro navázání spojení a jejím spuštění, kdy elektroměr odpovídal i po změně parametrů. Posledním nutným testem této etapy bylo zjištění, zda je možné pro navazování spojení používat vždy stejná aplikační data. Jak bylo zmíněno v kapitole 3.4.3, při navazování spojení přes software SmartSet se část jedné ze zpráv měnila podle nezjištěného pravidla. Opakované odesílání připojovacích zpráv prokázalo, že je možné používat vždy stejný obsah zpráv. Také bylo zjištěno, že spojení je nutné validně ukončit, protože na třetí pokus o připojení již elektroměr reaguje zamítavě. V tomto případě bylo nutné elektroměr na cca 10 sekund odpojit od napájení, což je možné provádět v laboratorních podmínkách, ale nikoliv v praxi.

3.6.2 Odečítání údaje o spotřebě

Protože již byla funkčnost samotné komunikace otestována v předchozí etapě, v této etapě byla pozornost soustředěna na korektní převod získané binární hodnoty na dekadickou. Nejdříve byly hodnoty čtyř bajtů reprezentující požadovanou hodnotu zobrazeny v GUI aplikace a ručním převodem získána jejich dekadická hodnota. Ta byla porovnána s hodnotou získanou softwarem SmartSet. Po zjištění, že se hodnoty shodují byla doprogramována část provádějící převod hodnoty zmíněných čtyřech bajtů na dekadickou, která je poté zobrazena v GUI aplikace.

3.6.3 Odpojení od elektroměru

Po zjištění o nutnosti validního ukončování spojení v první etapě bylo odpojení doprogramováno a následně otestováno. Po odpojení od elektroměru je možné aplikaci znovu spustit a měření zopakovat.

3.6.4 Webové API

Protože součástí práce není žádné uživatelské rozhraní pro zobrazení dat v databázi, byla funkcionalita ověřována pomocí webové aplikace PhpMyAdmin, která dovoluje procházet databázová data výpisem obsahu jednotlivých tabulek. Požadavky byly API předávány pomocí standardního webového prohlížeče zadání správné URL adresy. Prohlížeč poté zobrazil stromovou strukturu XML dokumentu, ve které byla provedena kontrola obsahu. V případě ukládání dat byl otestován požadavek s uloženým i neznámým Android ID. Následně bylo otestováno stažení údajů o zákazníkovi, opět s validním i neznámým ID. Po ověření funkčnosti byla naprogramována třída HttpReq a následně proběhlo její testování stejným způsobem. Nejdříve byla ověřena správnost sestavovaných URL adres, poté správnost získaného XML dokumentu a v poslední fázi jeho korektní parsování. Protože instance třídy HttpReq běží ve svých vláknech, bylo její chování nejdříve testováno pomocí emulátoru s odesíláním smyšlených dat. Až po ověření funkčnosti na emulátoru bylo pokračováno na tabletu.

4 ZÁVĚR

Tato práce se zabývala možností využít přenosná mobilní zařízení podporující funkci USB OTG k odečítání údajů z chytrých měřících zařízení podporujících protokol DLMS/COSEM. Dle zadání měla být v praktické části vyvinuta aplikace pro operační systém Android, která bude schopná komunikovat s elektroměrem společnosti Kaifa, odečítat z něj hodnotu naměřené spotřeby elektrické energie a tuto hodnotu následně uložit do vzdálené databáze. Tohoto cíle bylo v plném rozsahu dosaženo.

Před samotným vývojem aplikace bylo nutné seznámit se standardem DLMS/COSEM, především v jeho implementaci komunikace přes sériovou linku, která je vzhledem ke způsobu iniciace spojení značně nestandardní. Po nastudování této problematiky byla otestována kompatibilita celé testovací sady (tablet, optický senzor, elektroměr) naprogramováním části aplikace ošetřující navázání spojení s elektroměrem se všemi jeho kroky včetně implementačních odlišností od standardu. Tyto odlišnosti byly zjištěny analýzou zachycených dat mezi softwarem SmartSet a testovacím elektroměrem. Provést tuto analýzu bylo také nutné pro získání aplikačních dat, která nejsou součástí standardu a výrobce elektroměru je neposkytuje. Po ověření funkčnosti sériové linky byla aplikace dále rozšířena o další třídy pro komunikaci s elektroměrem včetně odečítání údaje o naměřené spotřebě a komunikaci s databází navrženou s ohledem na možné potřeby společnosti využívající prozkoumávanou metodu sběru dat. V databázi jsou kromě naměřených hodnot dále uchovávány údaje o zákaznících, zaměstnancích a jejich mobilních zařízeních. Databáze byla zprovozněna na vzdáleném serveru pomocí softwaru MariaDB, který s mobilní aplikací komunikuje prostřednictvím webového API, které bylo také naprogramováno jako součást této práce v jazyce PHP. Výstupem API je XML dokument, který je mobilní aplikace schopná parsovat a získaná data zobrazovat v grafickém uživatelském rozhraní, přes které uživatel ovládá všechny funkce aplikace.

Vytvořená aplikace by dále mohla být rozšířena o podporu dalších typů elektroměru, k čemuž je však nezbytné získat od daného výrobce nebo vlastním výzkumem aplikační data, která není možné získat ze standardu DLMS/COSEM protokolu. Další možností rozšíření je přidání podpory uložení naměřených hodnot s podporou dávkového odesílání do databáze. Tímto by byla ošetřena situace nedostupného internetového připojení v místě odečtu údajů nebo eliminována potřeba mobilního internetového připojení kompletně – získané hodnoty by byly uloženy až po připojení na Wi-Fi např. v centrále společnosti.

BIBLIOGRAFIE

[1] *Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015* [online]. 2015. [cit. 2016-12-10]. Dostupné z: http://www.gartner.com/newsroom/id/3165317

[2] HIGGINBOTHAM, Stacey. 2016. Prediction: there won't be 50B connected IoT devices by 2020. *Structure Connect* [online]. [cit. 2016-12-10]. Dostupné z: http://www.structureconnect.com/prediction-there-wont-be-50b-connected-iot-devices-by-2020/

[3] The History of M2M. *Connected World* [online]. 2009 [cit. 2016-10-07]. Dostupné z: https://connectedworld.com/the-history-of-m2m/

[4] HÖLLER, Jan. *From machine-to-machine to the Internet of things: introduction to a new age of intelligence* [online]. Amsterdam: Elsevier Academic Press, 2014 [cit. 2016-12-09]. ISBN 978-008-0994-017.

[5] BOSWARTHICK, David, Omar ELLOUMI a Olivier HERSENT. *M2M communications: a systems approach* [online]. Hoboken, N.J.: Wiley, 2012 [cit. 2016-12-09]. ISBN 9781119994756.

[6] Where the smart is. *The Economist* [online]. 2016 [cit. 2016-10-07]. Dostupné z: http://www.economist.com/news/business/21700380-connected-homes-will-take-longer-materialise-expected-where-smart

[7] CURRY, David. Will wind turbines and solar panels be IoT juice of choice? *ReadWrite* [online]. 2016 [cit. 2016-10-07]. Dostupné z: http://readwrite.com/2016/06/11/iot-smart-cities-renewables-pt4/

[8] NEAGLE, Colin. How Sigfox plans to spread its low-power IoT network across the U.S. *Network World* [online]. 2016 [cit. 2016-11-25]. Dostupné z: http://www.networkworld.com/article/3029253/internet-of-things/how-sigfox-plans-to-spread-its-low-power-iot-network-across-the-u-s.html

[9] *DLMS/COSEM: Architecture and Protocols*. 8. verze. DLMS User Association, 2015.

[10] *Remix OS* [online]. [cit. 2016-12-01]. Dostupné z: http://www.jide.com/remixos

[11] *MA110 Smart Meter User Manual* [online]. Shenzhen: SHENZHEN KAIFA TECHNOLOGY CO., 2014 [cit. 2016-12-06].

[12] Dashboards. *Android Developers* [online]. [cit. 2016-12-07]. Dostupné z: https://developer.android.com/about/dashboards/index.html

[13] *FelHR85/UsbSerial* [online]. GitHub [cit. 2016-11-25]. Dostupné z: https://github.com/felHR85/UsbSerial

[14] *MariaDB versus MySQL - Features* [online]. MariaDB [cit. 2016-11-14]. Dostupné z: https://mariadb.com/kb/en/mariadb/mariadb-vs-mysql-features/

[15] *Apache MPM Common Directives* [online]. [cit. 2016-11-14]. Dostupné z: http://httpd.apache.org/docs/2.4/mod/mpm_common.html#maxrequestworkers

[16] *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content* [online]. IETF [cit. 2016-11-10]. Dostupné z: https://tools.ietf.org/html/rfc7231

[17] *Sprintf* [online]. The PHP Group [cit. 2016-11-10]. Dostupné z: http://php.net/manual/en/function.sprintf.php

[18] *SQL Injection* [online]. Refsnes Data [cit. 2016-11-09]. Dostupné z: http://www.w3schools.com/Sql/sql_injection.asp

[19] *RFC 1662: HDLC-like Framing [online]. [cit. 2017-02-15]. Dostupné z: https://tools.ietf.org/html/rfc1662*

[20] *DLMS/COSEM: Architecture and Protocols [online]. 2015. Green Book Edition 8.1. DLMS User Association [cit. 2017-02-15].*

[21] *AsyncTask [online], [cit. 2017-04-30]. Dostupné z: https://developer.android.com/reference/android/os/AsyncTask.html*

[22] *Class DocumentBuilder, Java™ Platform Standard Ed. 7 [online]. [cit. 2017-05-09]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/javax/xml/parsers/DocumentBuilder.html*

[23] *Intent, Android Developers [online]. [cit. 2017-05-09]. Dostupné z: https://developer.android.com/reference/android/content/Intent.html*

[24] *LIST OF STANDARD OBIS CODES AND COSEM OBJECTS, DLMS User Association [online]. [cit. 2017-05-02]. Dostupné z: http://www.dlms.com/documentation/listofstandardobiscodesandmaintenanceproces/index.html*

[25] *Https://www.hhdsoftware.com/device-monitoring-studio [online], [cit. 2017-04-18]. Dostupné z: https://www.hhdsoftware.com/device-monitoring-studio*

SEZNAM ZKRATEK

ACK	Acknowledgement
API	Application Program Interface
CR	Carriage return
DB	Database, Databáze
HDLC	High-Level Data Link Control
HTML	HyperText Markup Language
IFF	Identify: friend or foe
IMEI	International Mobile Equipment Identity
IoT	Internet of Things
IP	Internet protocol
LF	Line feed
M2M	Machine to Machine
MSDU	MAC Service Data Unit
OBIS	Object Identification System
RFID	Radio frequency identification
SoC	Systém on a Chip
TCP	Transmission control protocol
URL	Uniform Resource Locator
XML	Extensible Markup Language

PŘÍLOHY

Zdrojové kódy

Navázání spojení s elektroměrem

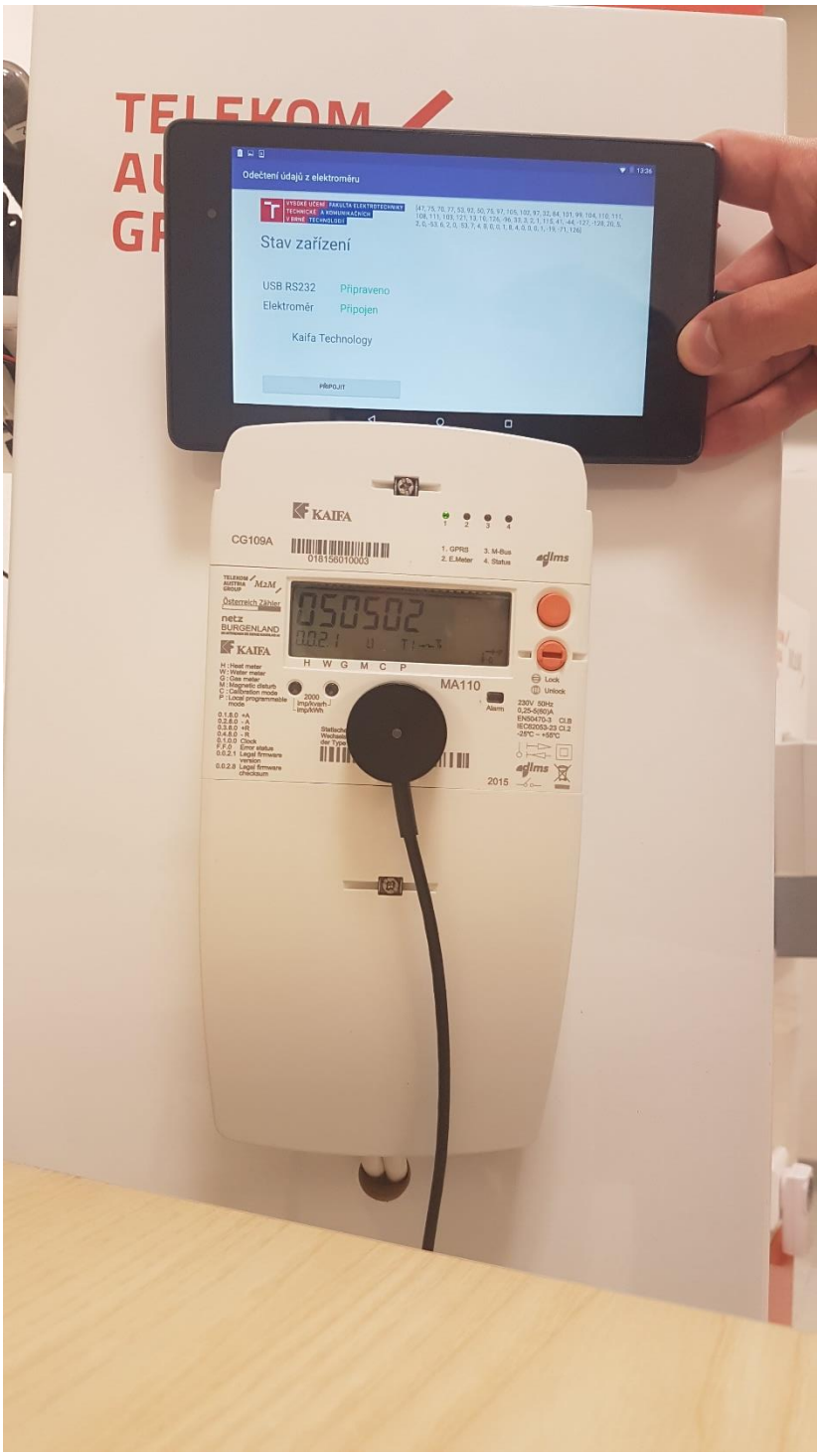
```
private class ElektromerConnection extends AsyncTask {

        private int action;

        public ElektromerConnection(int param) {

            action = param;
        }

  …
@Override
        protected Object doInBackground(Object[] params) {

            if (serialPort == null) return null;

            BUSY = true;

            switch (action) {
                //pripojeni k elmeru
                case ACTION_SENS_CON:

                    serialPort.setBaudRate(300);
                    serialPort.setDataBits(UsbSerialInterface.DATA_BITS_7);
                    serialPort.setStopBits(UsbSerialInterface.STOP_BITS_1);
                    serialPort.setParity(UsbSerialInterface.PARITY_EVEN);

serialPort.setFlowControl(UsbSerialInterface.FLOW_CONTROL_OFF);


                    sendData(hello1, 1300);
                    sendData(hello2, 500);


                    serialPort.setBaudRate(9600);
                    serialPort.setDataBits(UsbSerialInterface.DATA_BITS_8);
                    serialPort.setStopBits(UsbSerialInterface.STOP_BITS_1);
                    serialPort.setParity(UsbSerialInterface.PARITY_NONE);


                    sendData(hello3, 500);
                    sendData(hello4, 700);
                    sendData(hello5, 500);


                    break;
```

16bit CRC

```
class CRC16 {
    public CRC16() {

    }
    private static int[] lookupTable = { 0x0000, 0x1189, 0x2312, 0x329B, 0x4624,
            0x57AD, 0x6536, 0x74BF, 0x8C48, 0x9DC1, 0xAF5A, 0xBED3, 0xCA6C,
            0xDBE5, 0xE97E, 0xF8F7, 0x1081, 0x0108, 0x3393, 0x221A, 0x56A5,
            0x472C, 0x75B7, 0x643E, 0x9CC9, 0x8D40, 0xBFDB, 0xAE52, 0xDAED,
            0xCB64, 0xF9FF, 0xE876, 0x2102, 0x308B, 0x0210, 0x1399, 0x6726,
            0x76AF, 0x4434, 0x55BD, 0xAD4A, 0xBCC3, 0x8E58, 0x9FD1, 0xEB6E,
            0xFAE7, 0xC87C, 0xD9F5, 0x3183, 0x200A, 0x1291, 0x0318, 0x77A7,
            0x662E, 0x54B5, 0x453C, 0xBDCB, 0xAC42, 0x9ED9, 0x8F50, 0xFBEF,
            0xEA66, 0xD8FD, 0xC974, 0x4204, 0x538D, 0x6116, 0x709F, 0x0420,
            0x15A9, 0x2732, 0x36BB, 0xCE4C, 0xDFC5, 0xED5E, 0xFCD7, 0x8868,
            0x99E1, 0xAB7A, 0xBAF3, 0x5285, 0x430C, 0x7197, 0x601E, 0x14A1,
            0x0528, 0x37B3, 0x263A, 0xDECD, 0xCF44, 0xFDDF, 0xEC56, 0x98E9,
            0x8960, 0xBBFB, 0xAA72, 0x6306, 0x728F, 0x4014, 0x519D, 0x2522,
            0x34AB, 0x0630, 0x17B9, 0xEF4E, 0xFEC7, 0xCC5C, 0xDDD5, 0xA96A,
            0xB8E3, 0x8A78, 0x9BF1, 0x7387, 0x620E, 0x5095, 0x411C, 0x35A3,
            0x242A, 0x16B1, 0x0738, 0xFFCF, 0xEE46, 0xDCDD, 0xCD54, 0xB9EB,
            0xA862, 0x9AF9, 0x8B70, 0x8408, 0x9581, 0xA71A, 0xB693, 0xC22C,
            0xD3A5, 0xE13E, 0xF0B7, 0x0840, 0x19C9, 0x2B52, 0x3ADB, 0x4E64,
            0x5FED, 0x6D76, 0x7CFF, 0x9489, 0x8500, 0xB79B, 0xA612, 0xD2AD,
            0xC324, 0xF1BF, 0xE036, 0x18C1, 0x0948, 0x3BD3, 0x2A5A, 0x5EE5,
            0x4F6C, 0x7DF7, 0x6C7E, 0xA50A, 0xB483, 0x8618, 0x9791, 0xE32E,
            0xF2A7, 0xC03C, 0xD1B5, 0x2942, 0x38CB, 0x0A50, 0x1BD9, 0x6F66,
            0x7EEF, 0x4C74, 0x5DFD, 0xB58B, 0xA402, 0x9699, 0x8710, 0xF3AF,
            0xE226, 0xD0BD, 0xC134, 0x39C3, 0x284A, 0x1AD1, 0x0B58, 0x7FE7,
            0x6E6E, 0x5CF5, 0x4D7C, 0xC60C, 0xD785, 0xE51E, 0xF497, 0x8028,
            0x91A1, 0xA33A, 0xB2B3, 0x4A44, 0x5BCD, 0x6956, 0x78DF, 0x0C60,
            0x1DE9, 0x2F72, 0x3EFB, 0xD68D, 0xC704, 0xF59F, 0xE416, 0x90A9,
            0x8120, 0xB3BB, 0xA232, 0x5AC5, 0x4B4C, 0x79D7, 0x685E, 0x1CE1,
            0x0D68, 0x3FF3, 0x2E7A, 0xE70E, 0xF687, 0xC41C, 0xD595, 0xA12A,
            0xB0A3, 0x8238, 0x93B1, 0x6B46, 0x7ACF, 0x4854, 0x59DD, 0x2D62,
            0x3CEB, 0x0E70, 0x1FF9, 0xF78F, 0xE606, 0xD49D, 0xC514, 0xB1AB,
            0xA022, 0x92B9, 0x8330, 0x7BC7, 0x6A4E, 0x58D5, 0x495C, 0x3DE3,
            0x2C6A, 0x1EF1, 0x0F78

    };
    public int FCS(final byte[] buff, final int offset, final int count) {
        int fcs16 = 0xFFFF;
        for (int pos = offset; pos < offset + count; ++pos) {
            fcs16 = (int) (((fcs16 >> 8)
                    ^ lookupTable[(fcs16 ^ (int) buff[pos]) & 0xFF]) & 0xFFFF);
        }
        fcs16 = ~fcs16;
        fcs16 = ((fcs16 >> 8) & 0xFF) | (fcs16 << 8);
        return (fcs16 & 0xFFFF);
    }
}
```

Fotografie

Elektroměr s připevněným optickým senzorem



Legenda:	Odesílatel:
Shodné hodnoty	Elektroměr
Rozdílné hodnoty	SmartSet

Záznamy komunikace (2 zprávy ze 4)

2F	3F	21	0D	0A

2F	3F	21	0D	0A

2F	4B	46	4D	35	5C	32	4B	61	69	66	61	20	54	65	63
68	6E	6F	6C	6F	67	79	0D	0A							

2F	4B	46	4D	35	5C	32	4B	61	69	66	61	20	54	65	63
68	6E	6F	6C	6F	67	79	0D	0A							

6	32	35	32	0D	0A	7E	A0	21	2	1	3	93	48	F3	81
80	14	5	2	7	EE	6	2	7	EE	7	4	0	0	0	1
8	4	0	0	0	1	B5	D4	7E							

6	32	35	32	0D	0A	7E	A0	21	2	1	3	93	48	F3	81
80	14	5	2	7	EE	6	2	7	EE	7	4	0	0	0	1
8	4	0	0	0	1	B5	D4	7E							

7E	A0	21	3	2	1	73	29	D4	81	80	14	5	2	0	CB
6	2	0	CB	7	4	0	0	0	1	8	4	0	0	0	1
ED	B9	7E													

7E	A0	21	3	2	1	73	29	D4	81	80	14	5	2	0	CB
6	2	0	CB	7	4	0	0	0	1	8	4	0	0	0	1
ED	B9	7E													

7E	A0	59	2	1	3	10	8	0A	E6	E6	0	60	4A	A1	9
6	7	60	85	74	5	8	1	1	A6	0A	4	8	0	0	0
0	0	0	0	0	8A	2	7	80	8B	7	60	85	74	5	8
2	5	AC	12	80	10	C1	4A	CF	88	E5	5F	1D	FA	EF	92
1	19	96	D6	4D	D2	BE	10	4	0E	1	0	0	0	6	5F
1F	4	0	0	1F	1F	FF	FD	E1	77	7E					

7E	A0	59	2	1	3	10	8	0A	E6	E6	0	60	4A	A1	9
6	7	60	85	74	5	8	1	1	A6	0A	4	8	0	0	0
0	0	0	0	0	8A	2	7	80	8B	7	60	85	74	5	8
2	5	AC	12	80	10	45	4B	48	D5	8D	27	5E	7B	F1	2
75	BF	A7	7D	29	1A	BE	10	4	0E	1	0	0	0	6	5F
1F	4	0	0	1F	1F	FF	FD	5D	79	7E					

7E A0 65 3 2 1 30 84 48 E6 E7 0 61 56 A1 9
6 7 60 85 74 5 8 1 1 A2 3 2 1 0 A3 5
A1 3 2 1 0E 89 7 60 85 74 5 8 2 5 A4 0A
4 8 4B 46 4D 2 98 F3 34 13 88 2 7 80 AA 12
80 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 BE 10 4 0E 8 0 6 5F 1F 4 0 0 1A 1D
1 0 0 7 30 F9 7E

7E A0 65 3 2 1 30 84 48 E6 E7 0 61 56 A1 9
6 7 60 85 74 5 8 1 1 A2 3 2 1 0 A3 5
A1 3 2 1 0E 89 7 60 85 74 5 8 2 5 A4 0A
4 8 4B 46 4D 2 98 F3 34 13 88 2 7 80 AA 12
80 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 BE 10 4 0E 8 0 6 5F 1F 4 0 0 1A 1D
1 0 0 7 30 F9 7E

7E A0 2D 2 1 3 32 FB 30 E6 E6 0 C3 1 C1 0
0F 0 0 28 0 0 FF 1 1 9 11 10 1 23 45 67
FB EE 0D 82 E8 E2 62 72 F6 F7 EC 78 AA 4B 7E

7E A0 2D 2 1 3 32 FB 30 E6 E6 0 C3 1 C1 0
0F 0 0 28 0 0 FF 1 1 9 11 10 1 23 45 67
FB EE 0D 82 E8 E2 62 72 F6 F7 EC 78 AA 4B 7E

7E A0 26 3 2 1 52 7E D4 E6 E7 0 C7 1 C1 0
1 0 9 11 10 1 23 45 67 72 CA 0A D0 69 D3 8F
70 71 4E 3 E8 C7 BA 7E

7E A0 26 3 2 1 52 7E D4 E6 E7 0 C7 1 C1 0
1 0 9 11 10 1 23 45 67 2C AC E9 40 B3 48 1A
0 4D B0 E6 F3 A6 91 7E

7E A0 8 2 1 3 51 A3 27 7E

7E A0 8 3 2 1 51 CC E7 7E

7E A0 1A 2 1 3 54 C6 D2 E6 E6 0 C0 1 C1 0
3 1 0 1 8 0 FF 2 0 32 68 7E

7E A0 1A 2 1 3 54 C6 D2 E6 E6 0 C0 1 C1 0
3 1 0 1 8 0 FF 2 0 32 68 7E

```
7E A0 16 3  2  1  74 9B 44 E6 E7 0  C4 1  C1 0
6  0  2  1A 63 B9 94 7E
```

```
7E A0 16 3  2  1  74 9B 44 E6 E7 0  C4 1  C1 0
6  0  2  1A BB 7C CE 7E
```

```
7E A0 1A 2  1  3  76 D6 D0 E6 E6 0  C0 1  C1 0
3  1  0  1  8  0  FF 3  0  EA 71 7E
```

```
7E A0 1A 2  1  3  76 D6 D0 E6 E6 0  C0 1  C1 0
3  1  0  1  8  0  FF 3  0  EA 71 7E
```

```
7E A0 17 3  2  1  96 C3 8B E6 E7 0  C4 1  C1 0
2  2  0F 0  16 1E 61 D4 7E
```

```
7E A0 17 3  2  1  96 C3 8B E6 E7 0  C4 1  C1 0
2  2  0F 0  16 1E 61 D4 7E
```

```
7E A0 8  2  1  3  53 B1 4  7E
```

```
7E A0 8  2  1  3  53 B1 4  7E
```

```
7E A0 21 3  2  1  73 29 D4 81 80 14 5  2  0  CB
6  2  0  CB 7  4  0  0  0  1  8  4  0  0  0  1
ED B9 7E
```

```
7E A0 21 3  2  1  73 29 D4 81 80 14 5  2  0  CB
6  2  0  CB 7  4  0  0  0  1  8  4  0  0  0  1
ED B9 7E
```